Attorney's Docket No. 07539.120 PATENT

In The United States Patent And Trademark Office

In re:	Roy Hays		
Appl. No.:	09/782,685	Group Art Unit:	2155
Filed:	February 13, 2001	Examiner:	Tran, Philip B.
For:	METHOD AND SYSTEM FOR COLLECTING INFORMATION AT DISTRIBUTED LOCATIONS		

DECLARATION OF BILLY W. HENSLEY AND ROY HAYS UNDER 37 C.F.R. §§ 1.131

We, Billy W. Hensley and Roy Hay, being joint inventors of the claimed subject matter, do hereby declare and say as follows:

1. We are the joint inventors of the inventions claimed in the original and pending claims of the above-captioned patent application.

2. We have read and understand the above-captioned patent application, including the original specification and claims. We also have read and understand the Office Action dated March 6, 2006 and active claims 1-13.

3. We have read and understand the following art applied in the Office Action: U.S. Patent No. 6,692,436 to Bluth et al. (hereinafter "Bluth") filed on April 14, 2000.

4. We hereby incorporate by reference the CD-R filed on October 27, 2005. We submit that the creation date of each file contained on the CD-R and relied upon for overcoming the outstanding rejections predates the April 14, 2000 filing date of Bluth.

5. We hereby incorporate the report attached to my Declaration of November 22, 2004. The multiple entry dates subsequent to April 14, 2000 reflect dates on which files were

checked in or checked out from a database repository. Any modifications made to the files subsequent to April 14, 2000 were not necessary for or part of the first actual reduction to practice of the invention.

6. The software code set forth in the CD-R and existing prior to the April 14, 2000 filing date of Bluth constitutes an actual reduction to practice of the invention. We declare that the software code worked for its intended purpose and performed each and every function of claims 1-13.

7. We have attached a hardcopy print-out of the software code contained on the CD-R, as requested by the Examiner. The code has been annotated and line numbers added to assist the Examiner in identifying the relevant code for each claim feature. The top line of each page of the hardcopy print-out includes a directory path, with the file name corresponding to the "File(s)" set forth below in paragraphs 8-20. For example, for claim 1, phrase 1, the relevant code may be found at the hardcopy print-out with the path "C:\Documents and Settings\...\threadReceiver.cpp." at line 426 (on page 7 of the directory print-out).

8. With regard to claim 1, the following files existing prior to April 14, 2000 and submitted on the CD-R allowed claim 1 to be carried out prior to April 14, 2000:

providing user information for registered users, the user information comprising medical information specific to the registered users;

Module: LCKioskServer.exe

File(s): threadReceiver.h; threadReceiver.cpp

Method(s): CExportKiosks::buildDailyExport;

CExportLCUsers::buildDailyExport

Line Nos.: 426 *et seq.*

Comments: User information is packed into files and placed in FTP directory for kiosk pickup

receiving updates to the user information;

Module: LCKioskClient.exe

File(s): wndMonitorISP.h; wndMonitorISP.cpp

Method(s): onExchange; exchangeFiles; pullFtpFiles

Line Nos.: 197 *et seq.*, 681 *et seq.*, 804 *et seq.*

Comments: Kiosks pick up data update files via FTP

generating update user information;

Module: LCKioskServer.exe

File(s): threadReceiver.h; threadReceiver.cpp

Method(s): CExportKiosks::buildDailyExport
CExportLCUsers::buildDailyExport

Line Nos.: 416 *et seq.*

Comments: Server generates files for kiosks containing user information. The files are placed in a directory for pickup.

for each of the collection kiosks, receiving a request from the collection kiosk for the generated update user information;

Module: LCKioskClient.exe

File(s): threadReceiver.cpp; threadReceiver.h

Method(s): onTimerReceiveFiles

Line Nos.: 127 *et seq.*

Comments: All files are received via FTP. The LCKioskServer picks the files up in the receiver directory and processes them all.

sending to the requesting collection kiosk the update user information

Module: LCKioskServer.exe

File(s): threadReceiver.cpp; threadReceiver.h

Method(s): CExportKiosks:buildDailyExport;

CExportLCUsers::buildDailyExport

Line Nos.: 416 *et seq.*

Comments: Data from the server is stored on the kiosk in a local database. Logins on the kiosks are authenticated against database.

storing the update user information at the requesting collection kiosk for subsequent requests,

Module: LCKioskClient.exe

File(s): wndMonitorISP.h; wndMonitorISP.cpp

Line Nos.: 265 *et seq.*

wherein the collection kiosks use the update user information to verify whether a user is registered.

Module: LCKioskClient.exe

File(s): wndMonitorISP.h; wndMonitorISP.cpp

Line Nos.: 265 *et seq.*

9 With regard to claim 2, the following files existing prior to April 14, 2000 allowed claim 2 to be carried out prior to April 14, 2000:

wherein the collection kiosks operate as FTP clients and the computer system operates as an FTP server

Module: LCKioskClient.exe

File(s): wndMonitorISP.h; wndMonitorISP.cpp

Method(s): exchangeFiles; pullFtpFiles

Line Nos.: 681 *et seq.*, 804 *et seq.*

Comments: The server side FTP module is part of Windows operating system. The server generates files to be transferred and drops them into an FTP directory for kiosk pickup.

10. With regard to claim 3, the following files existing prior to April 14, 2000 allowed claim 3 to be carried out prior to April 14, 2000:

wherein the received update user information includes indications of whether to add a registered user, delete a registered user, or change information relating to a registered user

Module (server side): LCBroker.exe

File(s): xc_applyKioskTrans.ccp

Method(s): applyUsers

Line Nos.: 93 *et seq.*

Comments: Determines if a user can be applied as a new user or must be rejected, i.e., if user is already in system

Module (kiosk side): KCData.dll

File(s): CoKCData.h; CoKCdata.ccp

Method(s): getLCUser

Line Nos.: 184 *et seq.*

Comments: User's information has been updated with information from the server after data exchange. The user is in one of several statuses: a verified lifeclinic user, a candidate to become a lifeclinic user, or rejected by server.

11. With regard to claim 4, the following files existing prior to April 14, 2000 allowed claim 4 to be carried out prior to April 14, 2000:

wherein a collection kiosk sends a request for the generated update user information once a day

Module: LCKioskClient.exe

File(s): wndMonitorISP.h; wndMonitorISP.cpp

Method(s): onExchange; exchangeFiles; pullFtpFiles

Line Nos.: 197 *et seq.*, 681 *et seq.*, 804 *et seq.*

Comments: Kiosk pulls available updates

12. With regard to claim 5, the following files existing prior to April 14, 2000 allowed claim 5 to be carried out prior to April 14, 2000:

wherein the user information includes a user identifier and a password

Module: LCKioskClient.exe

File(s): wndMonitorISP.h; wndMonitorISP.cpp

Method(s): onExchange; exchangeFiles; exportData

Line Nos.: 197 *et seq.*, 681 *et seq.*, 898 *et seq.*

Module: KCData.dll

File(s): CoKCData.h; CoKCData.cpp

Method(s): getUnexportedData

Line Nos.: 396 *et seq.*

Comments: Data sent to the server contains requests to add new users, with request including login and password.

13. With regard to claim 6, the following files existing prior to April 14, 2000 allowed claim 6 to be carried out prior to April 14, 2000:

providing user information for registered users, the user information comprising medical information specific to the registered users;

Module: KCData.dll

File(s): CoKCUser.h; CoKCUser.cpp

Method(s): get_LCUser

Line Nos.: 184 *et seq.*

sending a request for updated user information;

Module: KioskClient.exe

File(s): wndMonitorISP.h; wndMonitorISP.cpp

Method(s): onExchange; exchangeFiles

Line Nos.: 197 *et seq.*, 681 *et seq.*

in response to sending the request, receiving the updated user information; and updating the provided user information for the registered user in accordance with the received updated user information so that the collection kiosk can verify whether a user of the collection kiosk is registered; and

Module: KioskClient.exe

File(s): wndMonitorISP.h; wndMonitorISP.cpp

Method(s): onExchange; exchangeFiles; pullFtpFiles

Line Nos.: 197 *et seq.*, 681 *et seq.*, 804 *et seq.*

Module: KCData.dll

File(s): CoKCData.h; CoKCData.cpp

Method(s): importLCUsers; getLCUser

Line Nos.: 627 *et seq.*, 184 *et seq.*

Comments: local user information updated with information from server via importLCUsers after data exchange.

storing the updated user information at the collection kiosk for subsequent requests

Module: LCKioskClient.exe

File(s): wndMonitorISP.h; wndMonitorISP.cpp

Line Nos.: 265 *et seq.*

14. With regard to claim 7, the following files existing prior to April 14, 2000 supported claim 7 prior to April 14, 2000:

a central computer system for a web site, the central computer system providing a repository for the information, registering users of the web site, and accessing the information; and

Module: LCKioskClient.exe

File(s): wndMonitorISP.h; wndMonitorISP.cpp

Line Nos.: 265 *et seq.*

Comments: The central computer is comprised of IIS, MS SQL, FTP, LCKioskServer, and LLCBroker.

Web Directories/Files:

i. \Member\Admin\Kiosk

- KioskCheck.asp; KioskSave.asp

ii. \Member\Admin\KioskAds

- dataAdmin.asp; default.asp; distlist.asp; distlog.asp distnew.asp; distsave.asp

iii. \Member\BloodPressure

- BloodPressure.asp; BloodPressure_4_2.asp; LoadBP.asp; LoadBP_4_2.asp; SaveBP.asp; SaveBP_4_2.asp

iv. \Member\Charts

- Chart.asp; EmailtoPhysician.asp; NormalBPRanges.asp; NormalCholesterolRAnges.asp; Review.asp; vitalchart.asp; NormalGlucoseRanges.asp

v. \Member\Login

- index.asp; Login.asp; loginError.asp; verifyUser.asp

vi. \Member\MemberInfo

- familyMember.asp; getSexCode.asp; LoadDependent.asp; loadMemberInfo.asp; MemberInfo.asp; MemberInfo_new.asp; SaveDependent.asp; saveMemberInfo.asp; saveMemberInfo_3_15.asp

vii. \Member\NewUser

- agreement.asp; agrmdecline.asp; Newuser.asp; Newuser_form2.asp; Welcome.asp

viii. \Member\Preferences

- ChangePassword.asp; emailUpdatePreference.asp; LoadUserPreference.asp; mainPagePreference.asp; newsPreference.asp; password.asp; Preferences.asp; SaveEmailUpdate.asp; SaveMainPagePreference.asp; SaveNewsPreference.asp; SavePreference.asp

ix. \Member\Pulse

- LoadPulse.asp; LoadPulse_4_2.asp; Pulse.asp Pulse_4_2.asp; SavePulse.asp; SavePulse_4_2.asp

x. \Member\Weight

- LoadWeight.asp; LoadWeight_4_2.asp; SaveWeight.asp; SaveWEight_4_2.asp; Weight.asp; Weight_4_2.asp

a plurality of collection kiosks for collecting information about users, for verifying whether a user is registered at the web site, and for sending the collected information to the central computer system when the user is registered.

Module: KioskClient.exe

File(s): wndMonitorISP.h; wndMonitorISP.cpp

Method(s): onExchange; exchangeFiles; pullFtpFiles

Line Nos.: 197 *et seq.*, 681 *et seq.*, 804 *et seq.*

Comments: Kiosks maintain local copy of centralized data

Module: KCData.dll

File(s): CoKCData.h; CoKCData.cpp

Method(s): getLCUser

Line Nos.: 184 *et seq.*

Comments: authenticates user against local database

15. With regard to claim 8, the following files existing prior to April 14, 2000 allowed claim 8 to be carried out prior to April 14, 2000:

wherein the information is medical information.

Module: KCData.DLL

File(s): CoBPReading.h; CoBPReading.cpp;

Line Nos.: 104 *et seq.*,

Module: KCData.DLL

File(s): CoWeightReading.h; CoWeightReading.cpp;

Line Nos.: 78 *et seq.*

16. With regard to claim 9, the following files existing prior to April 14, 2000 allowed claim 9 to be carried out prior to April 14, 2000:

registering the users at the web site when information about a user is collected at one of a plurality of collection kiosks,

Module: LCKioskServer.exe

File(s): threadReceiver.cpp; threadReceiver.h

Method(s): onTimerReceiveFiles

Line Nos.: 127 *et seq.*

Comments: all files received via FTP; LCKioskServer picks files up in the receive directory and processes through LCBroker.exe

Module: LCBroker.exe

File(s): xc_applyKioskTrans.cpp

Method(s): applyUsers

Line Nos.: 93 *et seq.*

determining whether the user is registered at the website; and when registered, sending the collected information to a computer system so that the collected information is accessible to the user through the web site.

Module: KCData.dll

File(s): CoKCUser.h; CoKCUser.cpp

Method(s): get_LCUser

Line Nos.: 227 *et seq.*

Module: KioskClient.exe

File(s): wndMonitorISP.h; wndMonitorISP.cpp

Method(s): onExchange; exchangeFiles

Line Nos.: 197 *et seq.*, 681 *et seq.*

Comments: Each kiosk authenticates users against local database during login. User statuses are verified user, candidate, and rejected. Non-users and rejected users become candidate and registration is scheduled to occur via subsequent data exchange with server.

17. With regard to claim 10, the following files existing prior to April 14, 2000 allowed claim 10 to be carried out prior to April 14, 2000:

wherein a collection kiosk automatically sends a request for the generated update user information periodically.

Module: LCKioskClient.exe

File(s): wndMonitorISP.h; wndMonitorISP.cpp

Line Nos.: 197 *et seq.*

18. With regard to claim 11, the following files existing prior to April 14, 2000 allowed claim 11 to be carried out prior to April 14, 2000:

wherein said sending a request for updated information is automatic and performed periodically.

Module: LCKioskClient.exe

File(s): wndMonitorISP.h; wndMonitorISP.cpp

Line Nos.: 235 *et seq.*

19. With regard to claim 12, the following files existing prior to April 14, 2000 allowed claim 12 to be carried out prior to April 14, 2000:

wherein said sending a request for updated information is automatic and performed daily

Module: LCKioskClient.exe

File(s): wndMonitorISP.h; wndMonitorISP.cpp

Line Nos.: 123 *et seq.*

20. With regard to claim 13, the following files existing prior to April 14, 2000 allowed claim 13 to be carried out prior to April 14, 2000:

the information comprising medical information specific to the registered users;

Module: LCKioskClient.exe

File(s): wndMonitorISP.h; wndMonitorISP.cpp

Line Nos.: 681 *et seq.*

the central computer system further is for receiving updates to the user information from the collection kiosks, generating update user information, and for each of the collection kiosks, receiving a request from the collection kiosk for the generated update user information and sending to the requesting collection kiosk the update user information.

Module: LCKioskClient.exe

File(s): wndMonitorISP.h; wndMonitorISP.cpp

Line Nos.: 681 *et seq.*

21. We hereby declare that all statements made herein of our knowledge are true and all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the above-captioned patent application or any patent issued thereon.

____________________________ ______________
Billy W. Hensley Date

[illegible] 25 Oct 2006
Roy Hays Date

```
// CoKCUser.cpp : Implementation of CKCUser
#include "stdafx.h"
#include "KCData.h"
#include "CoKCUser.h"
#include "CoKCData.h"
#include "CoBPReading.h"
#include "CoWeightReading.h"
#include "rs_kcdata.h"

////////////////////////////////////////////////////////////////////////////
// CKCUser

HRESULT CKCUser::FinalConstruct()
{
    m_pobjOwner = NULL;
    m_prsBP = NULL;
    m_prsWeight = NULL;

    m_vKioskUserID.vt = VT_NULL;
    m_vFirstName.vt = VT_NULL;
    m_vLastName.vt = VT_NULL;
    m_vMiddleName.vt = VT_NULL;
    m_vAddress1.vt = VT_NULL;
    m_vAddress2.vt = VT_NULL;
    m_vCity.vt = VT_NULL;
    m_vState.vt = VT_NULL;
    m_vZip.vt = VT_NULL;
    m_vPhone.vt = VT_NULL;
    m_vPassword.vt = VT_NULL;
    m_vEMail.vt = VT_NULL;
    m_vLCPassword.vt = VT_NULL;
    m_vLCUser.vt = VT_NULL;
    m_vUserStatus.vt = VT_NULL;

    return S_OK;
}

void CKCUser::FinalRelease()
{
    if (m_prsBP != NULL)
        delete m_prsBP;

    if (m_prsWeight != NULL)
        delete m_prsWeight;

    if (m_pobjOwner != NULL)
        m_pobjOwner->Release();

    return;
}

STDMETHODIMP CKCUser::InterfaceSupportsErrorInfo(REFIID riid)
{
    static const IID* arr[] =
    {
        &IID_IKCUser
    };
    for (int i=0; i < sizeof(arr) / sizeof(arr[0]); i++)
    {
        if (InlineIsEqualGUID(*arr[i],riid))
            return S_OK;
    }
    return S_FALSE;
}

STDMETHODIMP CKCUser::get_KioskUserID(VARIANT *pVal)
```

```
{
    *pVal = m_vKioskUserID;
    return S_OK;
}

STDMETHODIMP CKCUser::get_FirstName(VARIANT * pVal)
{
    VariantInit(pVal);
    *pVal = _variant_t(m_vFirstName).Detach();
    return S_OK;
}

STDMETHODIMP CKCUser::put_FirstName(VARIANT newVal)
{
    m_vFirstName = newVal;
    return S_OK;
}

STDMETHODIMP CKCUser::get_LastName(VARIANT *pVal)
{
    VariantInit(pVal);
    *pVal = _variant_t(m_vLastName).Detach();
    return S_OK;
}

STDMETHODIMP CKCUser::put_LastName(VARIANT newVal)
{
    m_vLastName = newVal;
    return S_OK;
}

STDMETHODIMP CKCUser::get_MiddleName(VARIANT *pVal)
{
    VariantInit(pVal);
    *pVal = _variant_t(m_vMiddleName).Detach();
    return S_OK;
}

STDMETHODIMP CKCUser::put_MiddleName(VARIANT newVal)
{
    m_vMiddleName = newVal;
    return S_OK;
}

STDMETHODIMP CKCUser::get_Address1(VARIANT *pVal)
{
    VariantInit(pVal);
    *pVal = _variant_t(m_vAddress1).Detach();
    return S_OK;
}

STDMETHODIMP CKCUser::put_Address1(VARIANT newVal)
{
    m_vAddress1 = newVal;
    return S_OK;
}

STDMETHODIMP CKCUser::get_Address2(VARIANT *pVal)
{
    VariantInit(pVal);
    *pVal = _variant_t(m_vAddress2).Detach();
    return S_OK;
}

STDMETHODIMP CKCUser::put_Address2(VARIANT newVal)
{
    m_vAddress2 = newVal;
    return S_OK;
}

STDMETHODIMP CKCUser::get_City(VARIANT *pVal)
{
    VariantInit(pVal);
    *pVal = _variant_t(m_vCity).Detach();
    return S_OK;
}

STDMETHODIMP CKCUser::put_City(VARIANT newVal)
{
    m_vCity = newVal;
    return S_OK;
}

STDMETHODIMP CKCUser::get_State(VARIANT *pVal)
{
    VariantInit(pVal);
    *pVal = _variant_t(m_vState).Detach();
    return S_OK;
}

STDMETHODIMP CKCUser::put_State(VARIANT newVal)
{
    m_vState = newVal;
    return S_OK;
}

STDMETHODIMP CKCUser::get_Zip(VARIANT *pVal)
{
    VariantInit(pVal);
    *pVal = _variant_t(m_vZip).Detach();
    return S_OK;
}

STDMETHODIMP CKCUser::put_Zip(VARIANT newVal)
{
    m_vZip = newVal;
    return S_OK;
}

STDMETHODIMP CKCUser::get_Phone(VARIANT *pVal)
{
    VariantInit(pVal);
    *pVal = _variant_t(m_vPhone).Detach();
    return S_OK;
}

STDMETHODIMP CKCUser::put_Phone(VARIANT newVal)
{
    m_vPhone = newVal;
    return S_OK;
}

STDMETHODIMP CKCUser::get_Password(VARIANT *pVal)
{
    VariantInit(pVal);
    *pVal = _variant_t(m_vPassword).Detach();
    return S_OK;
}

STDMETHODIMP CKCUser::put_Password(VARIANT newVal)
{
    m_vPassword = newVal;
    return S_OK;
}

STDMETHODIMP CKCUser::get_EMail(VARIANT *pVal)
{
    VariantInit(pVal);
    *pVal = _variant_t(m_vEMail).Detach();
    return S_OK;
}

STDMETHODIMP CKCUser::put_EMail(VARIANT newVal)
{
    m_vEMail = newVal;
    return S_OK;
}

STDMETHODIMP CKCUser::put_LCPassword(VARIANT newVal)
{
    m_vLCPassword = newVal;
    return S_OK;
}

STDMETHODIMP CKCUser::put_LCUser(VARIANT newVal)
{
    m_vLCUser = newVal;
    return S_OK;
}

STDMETHODIMP CKCUser::get_LCUser(VARIANT *pVal)
{
    VariantInit(pVal);
    *pVal = _variant_t(m_vLCUser).Detach();
    return S_OK;
}

STDMETHODIMP CKCUser::get_UserStatus(VARIANT *pVal)
{
    VariantInit(pVal);
    *pVal = _variant_t(m_vUserStatus).Detach();
    return S_OK;
}

STDMETHODIMP CKCUser::put_UserStatus(VARIANT newVal)
{
    m_vUserStatus = newVal;
    return S_OK;
}

STDMETHODIMP CKCUser::update()
{
    Crs_kcuser  rs;
    string strError;
    HRESULT hr = S_OK;

    rs.setActiveCommand("uptUser");
    rs.setParameter("kiosk_user_id", m_vKioskUserID);
    rs.setParameter("first_name", m_vFirstName);
    rs.setParameter("last_name", m_vLastName);
    rs.setParameter("middle_name", m_vMiddleName);
    rs.setParameter("address1", m_vAddress1);
    rs.setParameter("address2", m_vAddress2);
    rs.setParameter("city", m_vCity);
    rs.setParameter("state", m_vState);
    rs.setParameter("zip", m_vZip);
    rs.setParameter("email", m_vEMail);
    rs.setParameter("phone", m_vPhone);
    rs.setParameter("password", m_vPassword);
    rs.setParameter("user_status", m_vUserStatus);

    if (!m_pobjOwner->m_pconn->execute(rs))
    {
        m_pobjOwner->m_pconn->getLastError(strError);
        Error(strError.c_str(), IID_IKCUser, hr = E_FAIL);
    }

    return hr;
}

STDMETHODIMP CKCUser::addBPReading(VARIANT vSystolicBP, VARIANT vDiastolicBP, VARIANT ↙
    vPulse)
{
    HRESULT hr = S_OK;

    try
    {
        SYSTEMTIME tm;
        GetLocalTime(&tm);
        DATE dateNow;
        SystemTimeToVariantTime(&tm, &dateNow);

        Crs_blood_pressure      rs;

        rs.setActiveCommand("insNewReading");
        rs.setParameter("kiosk_id", _variant_t(m_pobjOwner->m_lKioskId));
        rs.setParameter("kiosk_user_id", m_vKioskUserID);
        rs.setParameter("reading_dt", _variant_t(dateNow));
        rs.setParameter("systolic_bp", _variant_t(vSystolicBP));
        rs.setParameter("diastolic_bp", _variant_t(vDiastolicBP));
        rs.setParameter("pulse", _variant_t(vPulse));

        if (!m_pobjOwner->m_pconn->execute(rs))
        {
            string strError;
            m_pobjOwner->m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }

        removeOldReadings(rs);
    }
    catch(_com_error & e)
    {
        Error((BSTR) e.Description(), IID_IKCUser, hr = e.Error());
    }
    catch(HRESULT hrError)
    {
        hr = hrError;
    }
    catch(...)
    {
        hr = E_FAIL;
    }

    return hr;
}

STDMETHODIMP CKCUser::addWeightReading(VARIANT vWeight)
{
    HRESULT hr = S_OK;

    try
    {
        SYSTEMTIME tm;
        GetLocalTime(&tm);
        DATE dateNow;
        SystemTimeToVariantTime(&tm, &dateNow);

        Crs_weight      rs;

        rs.setActiveCommand("insNewReading");
        rs.setParameter("kiosk_id", _variant_t(m_pobjOwner->m_lKioskId));
        rs.setParameter("kiosk_user_id", m_vKioskUserID);
        rs.setParameter("reading_dt", _variant_t(dateNow));
        rs.setParameter("weight", _variant_t(vWeight));

        if (!m_pobjOwner->m_pconn->execute(rs))
        {
            string strError;
            m_pobjOwner->m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }

        removeOldReadings(rs);
    }
    catch(_com_error & e)
    {
        Error((BSTR) e.Description(), IID_IKCUser, hr = e.Error());
    }
    catch(HRESULT hrError)
    {
        hr = hrError;
    }
    catch(...)
    {
        hr = E_FAIL;
    }

    return hr;
}


STDMETHODIMP CKCUser::addAlternateID(VARIANT vID, VARIANT vIdType)
{
    HRESULT hr = S_OK;

    try
    {
        Crs_alternate_id        rs;

        rs.setActiveCommand("insNewId");
        rs.setParameter("kiosk_id", _variant_t(m_pobjOwner->m_lKioskId));
        rs.setParameter("alternate_id", _variant_t(vID));
        rs.setParameter("kiosk_user_id", m_vKioskUserID);
        if (vIdType.vt == VT_EMPTY || vIdType.vt == VT_NULL)
            rs.setParameter("id_type", _variant_t(2L));
        else
            rs.setParameter("id_type", _variant_t(vIdType));

        if (!m_pobjOwner->m_pconn->execute(rs))
        {
            string strError;
            m_pobjOwner->m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }

    }
    catch(_com_error & e)
    {
        Error((BSTR) e.Description(), IID_IKCUser, hr = e.Error());
    }
    catch(HRESULT hrError)
    {
        hr = hrError;
    }
    catch(...)
    {
        hr = E_FAIL;
    }

    return hr;
}

STDMETHODIMP CKCUser::getFirstBP(VARIANT *pBPReading)
{
    HRESULT hr = S_OK;

    try
    {
        if (m_prsBP != NULL)
            delete m_prsBP;

        m_prsBP = new Crs_blood_pressure;
        m_prsBP->setActiveCommand("getReadings");
        m_prsBP->setParameter("kiosk_user_id", m_vKioskUserID);

        if (!m_pobjOwner->m_pconn->execute(*m_prsBP))
        {
            string strError;
            m_pobjOwner->m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCUser, hr = E_FAIL);
            throw E_FAIL;
        }

        hr = getNextBP(pBPReading);
    }
    catch(_com_error & e)
    {
        Error((BSTR) e.Description(), IID_IKCUser, hr = e.Error());
    }
    catch(HRESULT hrError)
    {
        hr = hrError;
    }
    catch(...)
    {
        hr = E_FAIL;
    }

    return hr;
}

STDMETHODIMP CKCUser::getNextBP(VARIANT *pvBPReading)
{
    HRESULT hr = S_OK;
    VariantClear(pvBPReading);
    CComObject<CBPReading> * pobjBP = NULL;
    _variant_t vVal((IDispatch *) NULL, false);

    try
    {
        if (m_prsBP == NULL)
        {
            Error("CKCUser::getFirstBP() must be called first", IID_IKCUser, E_FAIL);
            throw E_FAIL;
        }

        if (!m_prsBP->isEOF())
        {
            if (FAILED(hr = CComObject<CBPReading>::CreateInstance(&pobjBP)))
            {
                stringstream strmError;
                strmError << "CComObject<CBPReading>::CreateInstance() failed. Error = [0x";
                strmError << std::hex << hr << "]";
                Error(strmError.str().c_str(), IID_IKCUser, hr);
                throw hr;
            }

            pobjBP->load(*m_prsBP);
            m_prsBP->moveNext();

            IDispatch * pIDispatch;
            if (FAILED(hr = pobjBP->QueryInterface(IID_IDispatch, (void **) & pIDispatch)))
            {
                stringstream strmError;
                strmError << "QueryInterface(IDispatch) failed. Error = [0x";
                strmError << std::hex << hr << "]";
                Error(strmError.str().c_str(), IID_IKCUser, hr);
                pobjBP->Release();
                throw hr;
            }

            vVal.pdispVal = pIDispatch;
        }
    }
    catch(_com_error & e)
    {
        Error((BSTR) e.Description(), IID_IKCUser, hr = e.Error());
    }
    catch(HRESULT hrError)
    {
        hr = hrError;
    }
    catch(...)
    {
        hr = E_FAIL;
    }

    if (SUCCEEDED(hr))
        *pvBPReading = vVal.Detach();

    return hr;
}

STDMETHODIMP CKCUser::getFirstWeight(VARIANT * pvWeight)
{
    HRESULT hr = S_OK;

    try
    {
        if (m_prsWeight != NULL)
            delete m_prsWeight;

        m_prsWeight = new Crs_weight;
        m_prsWeight->setActiveCommand("getReadings");
        m_prsWeight->setParameter("kiosk_user_id", m_vKioskUserID);

        if (!m_pobjOwner->m_pconn->execute(*m_prsWeight))
        {
            string strError;
            m_pobjOwner->m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCUser, hr = E_FAIL);
            throw E_FAIL;
        }

        hr = getNextWeight(pvWeight);
    }
    catch(_com_error & e)
    {
        Error((BSTR) e.Description(), IID_IKCUser, hr = e.Error());
    }
    catch(HRESULT hrError)
    {
        hr = hrError;
    }
    catch(...)
    {
        hr = E_FAIL;
    }

    return hr;
}

STDMETHODIMP CKCUser::getNextWeight(VARIANT *pvWeight)
{
    HRESULT hr = S_OK;
    VariantClear(pvWeight);
    CComObject<CWeightReading> * pobjWeight = NULL;
    _variant_t vVal((IDispatch *) NULL, false);

    try
    {
        if (m_prsWeight == NULL)
        {
            Error("CKCUser::getFirstWeight() must be called first", IID_IKCUser, E_FAIL);
            throw E_FAIL;
        }

        if (!m_prsWeight->isEOF())
        {
            if (FAILED(hr = CComObject<CWeightReading>::CreateInstance(&pobjWeight)))
            {
                stringstream strmError;
                strmError << "CComObject<CWeightReading>::CreateInstance() failed. Error = [0x";
                strmError << std::hex << hr << "]";
                Error(strmError.str().c_str(), IID_IKCUser, hr);
                throw hr;
            }

            pobjWeight->load(*m_prsWeight);
            m_prsWeight->moveNext();

            IDispatch * pIDispatch;
            if (FAILED(hr = pobjWeight->QueryInterface(IID_IDispatch, (void **) & pIDispatch)))
            {
                stringstream strmError;
                strmError << "QueryInterface(IDispatch) failed. Error = [0x";
                strmError << std::hex << hr << "]";
                Error(strmError.str().c_str(), IID_IKCUser, hr);
                pobjWeight->Release();
                throw hr;
            }

            vVal.pdispVal = pIDispatch;
        }
    }
    catch(_com_error & e)
    {
        Error((BSTR) e.Description(), IID_IKCUser, hr = e.Error());
    }
    catch(HRESULT hrError)
    {
        hr = hrError;
    }
    catch(...)
    {
        hr = E_FAIL;
    }

    if (SUCCEEDED(hr))
        *pvWeight = vVal.Detach();

    return hr;
}

STDMETHODIMP CKCUser::validatePassword(VARIANT vPassword, VARIANT vPWordType, VARIANT ↙
    *pvfValid)
{
    HRESULT hr = S_OK;
    VariantInit(pvfValid);

    try
    {
        pvfValid->vt = VT_BOOL;
        pvfValid->bVal = 0;

        long lPWordType;
        if (vPWordType.vt == VT_EMPTY || vPWordType.vt == VT_NULL)
            lPWordType = 1;
        else
            lPWordType = (long) _variant_t(vPWordType);

        if (lPWordType != 1 && lPWordType != 2)
        {
            Error("Invalid PasswordType.", IID_IKCUser, E_INVALIDARG);
            hr = E_INVALIDARG;
        }
        else
        {
            _bstr_t bstrPWordIn(vPassword);
            _bstr_t bstrPWord;
            if (lPWordType == 1)
                bstrPWord = m_vLCPassword;
            else
                bstrPWord = m_vPassword;

            string strBPWordIn;
            string strPWord;

            if (bstrPWordIn.length())
                strBPWordIn = (char *) bstrPWordIn;

            if (bstrPWord.length())
                strPWord = (char *) bstrPWord;

            TOUPPER(strBPWordIn);
            TOUPPER(strPWord);

            if (strBPWordIn.compare(strPWord) == 0)
                pvfValid->boolVal = -1;
            else
                pvfValid->boolVal = 0;
        }
    }
    catch(_com_error & e)
    {
        Error((BSTR) e.Description(), IID_IKCUser, hr = e.Error());
    }
    catch(...)
    {
        Error("Unknown exception", IID_IKCUser, hr = E_FAIL);
    }

    return hr;
}

////////////////////////////////////////////////////////////////////////////////
// internal C++ interface

bool CKCUser::load(CSdoRecordset & rs)
{
    m_vKioskUserID = rs.getField("kiosk_user_id");
    m_vFirstName = rs.getField("first_name");
    m_vLastName = rs.getField("last_name");
    m_vMiddleName = rs.getField("middle_name");
    m_vAddress1 = rs.getField("address1");
    m_vAddress2 = rs.getField("address2");
    m_vCity = rs.getField("city");
    m_vState = rs.getField("state");
    m_vZip = rs.getField("zip");
    m_vPhone = rs.getField("phone");
    m_vPassword = rs.getField("password");
    m_vUserStatus = rs.getField("user_status");

    return true;
}

void CKCUser::setOwner(CKCData * pobjOwner)
{
    m_pobjOwner = pobjOwner;
    m_pobjOwner->AddRef();
    return;
}

void CKCUser::removeOldReadings(CSdoRecordset & rs)
{
    _variant_t vLastReading;
    int nNumber = 0;
    string strError;

    rs.setActiveCommand("getReadings");
    rs.setParameter("kiosk_user_id", m_vKioskUserID);
    if (!m_pobjOwner->m_pconn->execute(rs))
    {
        m_pobjOwner->m_pconn->getLastError(strError);
        Error(strError.c_str(), IID_IKCData, E_FAIL);
        throw E_FAIL;
    }

    while (!rs.isEOF())
    {
        nNumber++;
        if (nNumber >= m_pobjOwner->m_lReadingsToKeep)
        {
            vLastReading = rs.getField("reading_dt");
            break;
        }
        rs.moveNext();
    }

    rs.close();

    if (vLastReading.vt != VT_EMPTY)
    {
        rs.setActiveCommand("removeOld");
        rs.setParameter("reading_dt", vLastReading);
        if (!m_pobjOwner->m_pconn->execute(rs))
        {
            m_pobjOwner->m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }
    }

    return;
}
```

```
// CoBPReading.cpp : Implementation of CBPReading
#include "stdafx.h"
#include "KCData.h"
#include "CoBPReading.h"
#include "rs_kcdata.h"

//////////////////////////////////////////////////////////////////////////////
// CBPReading

STDMETHODIMP CBPReading::InterfaceSupportsErrorInfo(REFIID riid)
{
    static const IID* arr[] =
    {
        &IID_IBPReading
    };
    for (int i=0; i < sizeof(arr) / sizeof(arr[0]); i++)
    {
        if (InlineIsEqualGUID(*arr[i],riid))
            return S_OK;
    }
    return S_FALSE;
}

HRESULT CBPReading::FinalConstruct()
{
    m_vPulse.vt = VT_NULL;
    m_vDiastolicBP.vt = VT_NULL;
    m_vSystolicBP.vt = VT_NULL;
    m_vReadingDt.vt = VT_NULL;
    m_vKioskUserID.vt = VT_NULL;
    return S_OK;
}

void CBPReading::FinalRelease()
{
}

STDMETHODIMP CBPReading::get_KioskUserID(VARIANT *pVal)
{
    _variant_t vVal(*pVal, false);
    vVal = m_vKioskUserID;
    *pVal = vVal.Detach();
    return S_OK;
}

STDMETHODIMP CBPReading::get_ReadingDate(VARIANT *pVal)
{
    _variant_t vVal(*pVal, false);
    vVal = m_vReadingDt;
    *pVal = vVal.Detach();
    return S_OK;
}

STDMETHODIMP CBPReading::put_ReadingDate(VARIANT newVal)
{
    m_vReadingDt = newVal;
    return S_OK;
}

STDMETHODIMP CBPReading::get_SystolicBP(VARIANT *pVal)
{
    _variant_t vVal(*pVal, false);
    vVal = m_vSystolicBP;
    *pVal = vVal.Detach();
    return S_OK;
}
```

```
STDMETHODIMP CBPReading::put_SystolicBP(VARIANT newVal)
{
    m_vSystolicBP = newVal;
    return S_OK;
}

STDMETHODIMP CBPReading::get_DiastolicBP(VARIANT *pVal)
{
    _variant_t vVal(*pVal, false);
    vVal = m_vDiastolicBP;
    *pVal = vVal.Detach();
    return S_OK;
}

STDMETHODIMP CBPReading::put_DiastolicBP(VARIANT newVal)
{
    m_vDiastolicBP = newVal;
    return S_OK;
}

STDMETHODIMP CBPReading::get_Pulse(VARIANT *pVal)
{
    _variant_t vVal(*pVal, false);
    vVal = m_vPulse;
    *pVal = vVal.Detach();
    return S_OK;
}

STDMETHODIMP CBPReading::put_Pulse(VARIANT newVal)
{
    m_vPulse = newVal;
    return S_OK;
}

/////////////////////////////////////////////////////////////////////////////
// internal C++ interface
bool CBPReading::load(CSdoRecordset & rs)
{
    m_vPulse = rs.getField("pulse");
    m_vDiastolicBP = rs.getField("diastolic_bp");
    m_vSystolicBP = rs.getField("systolic_bp");
    m_vReadingDt = rs.getField("reading_dt");
    m_vKioskUserID = rs.getField("kiosk_user_id");
    return true;
}


```

```
// CoKCData.cpp : Implementation of CKCData
#include "stdafx.h"
#include "KCData.h"
#include "CoKCData.h"
#include "CoKCUser.h"
#include "registryKCData.h"
#include "Encryptor.h"

#include "rs_kcdata.h"

HRESULT CKCData::FinalConstruct()
{
    m_lKioskId = 0;
    m_pconn = NULL;
    return CoCreateFreeThreadedMarshaler(
        GetControllingUnknown(), &m_pUnkMarshaler.p);
}

void CKCData::FinalRelease()
{
    close();
    m_pUnkMarshaler.Release();
}


/////////////////////////////////////////////////////////////////////////////
// CKCData

STDMETHODIMP CKCData::InterfaceSupportsErrorInfo(REFIID riid)
{
    static const IID* arr[] =
    {
        &IID_IKCData
    };

    for (int i=0; i < sizeof(arr) / sizeof(arr[0]); i++)
    {
        if (InlineIsEqualGUID(*arr[i],riid))
            return S_OK;
    }

    return S_FALSE;
}

STDMETHODIMP CKCData::open()
{
    HRESULT hr = S_OK;

    try
    {
        CRegistryKCData registry;
        if (!registry.open())
        {
            Error(registry.m_strLastError.c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }

        m_lKioskId = registry.m_lKioskId;
        m_lReadingsToKeep = registry.m_lNumberReadingsToKeep;

        if (m_pconn != NULL)
            delete m_pconn;

        m_pconn = new CSdoConnection;

        // these options are not compatible with access driver
        m_pconn->setCommitOptionsOnConnect(false);
        m_pconn->useProvider(false);
        m_pconn->setForwardRecordsetToOpen(false);

        string strConn = "DSN=";
        strConn += registry.m_strKCDataSource + ";";

        if (!m_pconn->connect(  strConn.c_str(),
                                registry.m_strKCUser.c_str(),
                                registry.m_strKCPassword.c_str() ))
        {
            string strError;
            m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }
    }
    catch(HRESULT hrError)
    {
        hr = hrError;
    }
    catch(...)
    {
        hr = E_FAIL;
    }

    return hr;
}

STDMETHODIMP CKCData::close()
{
    if (m_pconn)
    {
        m_pconn->close();
        delete m_pconn;
    }

    m_pconn = NULL;
    return S_OK;
}

STDMETHODIMP CKCData::buildRegistry()
{
    HRESULT hr = S_OK;

    CRegistryKCData  registry;
    if (!registry.buildInitial())
        Error(registry.m_strLastError.c_str(), IID_IKCData, hr = E_FAIL);

    return hr;
}

STDMETHODIMP CKCData::getKioskUser(VARIANT vKioskId, VARIANT vIdType, VARIANT *
    pvDispUser)
{
    VariantInit(pvDispUser);
    _variant_t vRetVal((IDispatch *) NULL, false);

    HRESULT hr = S_OK;

    try
    {
        CRs_KCData rs;
        rs.setActiveCommand("cmdGetById");
        rs.setParameter("alternate_id", _variant_t(vKioskId));
        rs.setParameter("id_type", _variant_t(vIdType));
        if (!m_pconn->execute(rs))
        {
            string strError;
            m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, hr = E_FAIL);
            throw hr;
        }

        if (!rs.isEmpty())
        {
            CComObject<CKCUser> * pobjUser;

            if (FAILED(hr = CComObject<CKCUser>::CreateInstance(&pobjUser)))
            {
                Error("CComObject<CKUser>::CreateInstance() failed.", IID_IKCUser, hr)↙
;
                throw hr;
            }

            pobjUser->setOwner(this);
            IDispatch * pIDisp = NULL;

            if (FAILED(hr = pobjUser->QueryInterface(IID_IDispatch, (void **) &        ↙
    pIDisp)))
            {
                Error("QueryInterface(IDispatch **) failed", IID_IKCUser, hr);
                pobjUser->Release();
                throw hr;
            }

            vRetVal = pIDisp;

            pobjUser->load(rs);
        }
    }
    catch(_com_error & e)
    {
        _bstr_t bstrError = e.Description();
        Error((char *) bstrError, IID_IKCData, hr = e.Error());
    }
    catch(HRESULT hrError)
    {
        hr = hrError;
    }
    catch(...)
    {
        hr = E_FAIL;
    }

    *pvDispUser = vRetVal;

    return hr;
}

STDMETHODIMP CKCData::getLCUser(VARIANT vLogon, VARIANT *pvUser)
{
    VariantInit(pvUser);

    _variant_t vRetVal((IDispatch *) NULL, false);

    HRESULT hr = S_OK;

    try
    {
        // check to see if LC user
        Crs_lifeclinic_users rsLC;
        rsLC.setActiveCommand("getByName");
        rsLC.setParameter("user_name", _variant_t(vLogon));
        if (!m_pconn->execute(rsLC))
        {
            string strError;
            m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, hr = E_FAIL);
            throw hr;
        }

        _variant_t vUserName;
        _variant_t vUserPassword;
        _variant_t vCpiId;
        _variant_t vUser;

        if (!rsLC.isEmpty())
        {
            vCpiId = rsLC.getField("lifeclinic_id");
            string strEncryptedPassword;
            string strPassword;
            rsLC.getField("password", strEncryptedPassword);
            if (strEncryptedPassword.size())
            {
                CEncryptor encryptor;
                encryptor.Decrypt(strEncryptedPassword.c_str(), NULL, strPassword);
                vUserPassword = strPassword.c_str();
            }
            vUserName = rsLC.getField("user_name");
            rsLC.close();

            if (FAILED(getKioskUser(vCpiId, _variant_t((long) IdType_LC), &vRetVal)))
                throw E_FAIL;

            IKCUser * pIKCUser = NULL;
            bool fNewUser = false;
            if (vRetVal.pdispVal == NULL)
            {
                if (FAILED(createUser(&vRetVal)))
                    throw E_FAIL;

                fNewUser = true;
            }

            if (FAILED(hr = vRetVal.pdispVal->QueryInterface(IID_IKCUser, (void **) & ✔
    pIKCUser)))
            {
                stringstream strmError;
                strmError << "pIDispatch->QueryInterface(IID_IKCuser) failed. Error = ✔
    {";
                strmError << std::hex << hr << "}.";
                Error(strmError.str().c_str(), IID_IKCData, hr);
                throw hr;
            }

            if (fNewUser)
            {
                pIKCUser->addAlternateID(vCpiId, _variant_t((long) IdType_LC));
                pIKCUser->put_UserStatus(_variant_t((long) UStat_LCnewKC));
                pIKCUser->update();
            }

            pIKCUser->put_LCUser(vUserName);
            pIKCUser->put_LCPassword(vUserPassword);
            pIKCUser->Release();
        }
    }
    catch(_com_error & e)
    {
        _bstr_t bstrError = e.Description();
        Error((char *) bstrError, IID_IKCData, hr = e.Error());
    }
    catch(HRESULT hrError)
    {
        hr = hrError;
    }
    catch(...)
    {
        hr = E_FAIL;
    }

    *pvUser = vRetVal.Detach();

    return hr;
}


STDMETHODIMP CKCData::createUser(VARIANT * pvUser)
{
    HRESULT hr = S_OK;
    bool fTransBegun = false;
    string strError;
    CComObject<CKCUser> * pobjUser = NULL;

    try
    {
        if (m_pconn == NULL || !m_pconn->isConnected())
        {
            Error("There is no database connection", IID_IKCData, hr = E_FAIL);
            throw hr;
        }

        ////////////////////////////////////////////////////////////////////
        // create underlying database record
        fTransBegun = m_pconn->beginTrans();

        Crs_kcuser      rs;
        rs.setActiveCommand("cmdInsNewUser");
        rs.setParameter("kiosk_id", _variant_t(m_lKioskId));
        if (!m_pconn->execute(rs))
        {
            m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }

        // get the newly created id
        rs.setActiveCommand("cmdGetNewUser");
        if (!m_pconn->execute(rs))
        {
            m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }
        _variant_t vNewId = rs.getField("kiosk_user_id");
        rs.close();

        // mark record as in use
        rs.setActiveCommand("setUserInUse");
        rs.setParameter("in_use", _variant_t(-1L));
        rs.setParameter("kiosk_user_id", vNewId);
        if (!m_pconn->execute(rs))
        {
            m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }

        // put primary id in id map
        rs.setActiveCommand("putIdMap");
        rs.setParameter("kiosk_id", _variant_t(m_lKioskId));
        rs.setParameter("alternate_id", vNewId);
        rs.setParameter("kiosk_user_id", vNewId);
        if (!m_pconn->execute(rs))
        {
            m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }

        //////////////////////////////////////////////////////////////////
        // create com object
        if (FAILED(hr = CComObject<CKCUser>::CreateInstance(&pobjUser)))
        {
            Error("CComObject<CKCUser>::CreateInstance() failed", IID_IKCData, hr);
            throw hr;
        }
        pobjUser->setOwner(this);
        pobjUser->m_vKioskUserID = vNewId;


        // return IDispatch in VARIANT
        IDispatch * pIDisp = NULL;
        if (FAILED(hr = pobjUser->QueryInterface(IID_IDispatch, (void**) &pIDisp)))
        {
            Error("CComObject<CKCUser>::QueryInterface(IDispatch) failed", IID_IKCData
    , hr);
            throw hr;
        }

        _variant_t vRet(pIDisp, false);
        *pvUser = vRet.Detach();

        pobjUser = NULL;
    }
    catch(_com_error & e)
    {
        _bstr_t bstrError = e.Description();
        Error((char *) bstrError, IID_IKCData, hr = e.Error());
    }
    catch(HRESULT hrError)
    {
        hr = hrError;
    }
    catch(...)
    {
        Error("Unknown exception", IID_IKCData, hr = E_FAIL);
    }

    if (fTransBegun)
    {
        if (SUCCEEDED(hr))
            m_pconn->commitTrans();
        else
            m_pconn->rollbackTrans();
    }

    // if pobjUser is valued then an error ocurred after its creation
    if (pobjUser != NULL)
```

```
        delete pobjUser;

    return hr;
}

STDMETHODIMP CKCData::getUnexportedData(VARIANT *pvData)
{
    HRESULT hr = S_OK;

    VariantInit(pvData);
    string strError;

    try
    {
        Crs_kcuser rsUser;
        Crs_blood_pressure rsBP;
        Crs_weight rsWeight;
        Crs_alternate_id rsIds;

        rsUser.setActiveCommand("getUnExported");
        rsBP.setActiveCommand("getUnExported");
        rsWeight.setActiveCommand("getUnExported");
        rsIds.setActiveCommand("getUnExported");

        if (!m_pconn->execute(rsUser))
        {
            m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }

        if (!m_pconn->execute(rsBP))
        {
            m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }

        if (!m_pconn->execute(rsWeight))
        {
            m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }

        if (!m_pconn->execute(rsIds))
        {
            m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }

        CXmlDocument xdocXData("<kiosk_data/>");
        xdocXData.setUpperCaseTags(false);

        rsUser.toXml(xdocXData);
        rsBP.toXml(xdocXData);
        rsWeight.toXml(xdocXData);
        rsIds.toXml(xdocXData);

        string strXml;
        xdocXData.getXML(strXml);

        _variant_t vVal = strXml.c_str();
        *pvData = vVal.Detach();
    }
    catch(_com_error & e)
    {
        Error((BSTR) e.Description(), IID_IKCData, hr = e.Error());
    }
    catch(HRESULT hrError)
    {
        hr = hrError;
    }
    catch(...)
    {
        hr = E_FAIL;
    }

    return hr;
}

STDMETHODIMP CKCData::markDataExported()
{
    HRESULT hr = S_OK;
    string strError;
    bool fTransStarted = false;

    try
    {
        Crs_kcuser rsUser;
        Crs_blood_pressure rsBP;
        Crs_weight rsWeight;
        Crs_alternate_id rsIds;

        rsUser.setActiveCommand("markExported");
        rsBP.setActiveCommand("markExported");
        rsWeight.setActiveCommand("markExported");
        rsIds.setActiveCommand("markExported");

        if (!(fTransStarted = m_pconn->beginTrans()))
        {
            m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }

        if (!m_pconn->execute(rsUser))
        {
            m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }

        if (!m_pconn->execute(rsBP))
        {
            m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }

        if (!m_pconn->execute(rsWeight))
        {
            m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }

        if (!m_pconn->execute(rsIds))
        {
            m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, E_FAIL);
```

```
                throw E_FAIL;
            }
        }
        catch(_com_error & e)
        {
            Error((BSTR) e.Description(), IID_IKCData, hr = e.Error());
        }
        catch(HRESULT hrError)
        {
            hr = hrError;
        }
        catch(...)
        {
            hr = E_FAIL;
        }

        if (fTransStarted)
        {
            if (SUCCEEDED(hr))
                m_pconn->commitTrans();
            else
                m_pconn->rollbackTrans();
        }

        return hr;
}

STDMETHODIMP CKCData::markDataUnexported()
{
        HRESULT hr = S_OK;
        string strError;
        bool fTransStarted = false;

        try
        {
            Crs_kcuser rsUser;
            Crs_blood_pressure rsBP;
            Crs_weight rsWeight;
            Crs_alternate_id rsIds;

            rsUser.setActiveCommand("markUnexported");
            rsBP.setActiveCommand("markUnexported");
            rsWeight.setActiveCommand("markUnexported");
            rsIds.setActiveCommand("markUnexported");

            if (!(fTransStarted = m_pconn->beginTrans()))
            {
                m_pconn->getLastError(strError);
                Error(strError.c_str(), IID_IKCData, E_FAIL);
                throw E_FAIL;
            }

            if (!m_pconn->execute(rsUser))
            {
                m_pconn->getLastError(strError);
                Error(strError.c_str(), IID_IKCData, E_FAIL);
                throw E_FAIL;
            }

            if (!m_pconn->execute(rsBP))
            {
                m_pconn->getLastError(strError);
                Error(strError.c_str(), IID_IKCData, E_FAIL);
                throw E_FAIL;
            }
        if (!m_pconn->execute(rsWeight))
        {
            m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }

        if (!m_pconn->execute(rsIds))
        {
            m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }
    }
    catch(_com_error & e)
    {
        Error((BSTR) e.Description(), IID_IKCData, hr = e.Error());
    }
    catch(HRESULT hrError)
    {
        hr = hrError;
    }
    catch(...)
    {
        hr = E_FAIL;
    }

    if (fTransStarted)
    {
        if (SUCCEEDED(hr))
            m_pconn->commitTrans();
        else
            m_pconn->rollbackTrans();
    }

    return hr;
}

STDMETHODIMP CKCData::importLCUsers(VARIANT vFileName)
{
    HRESULT hr = S_OK;

    try
    {
        string strFileName = (char *) (_bstr_t) vFileName;

        CXmlDocument xdocLCUsers;
        if (!loadXmlFile(strFileName.c_str(), xdocLCUsers, false))
            throw E_FAIL;

        CXmlElement elTable;

        if (xdocLCUsers.getItem("cpi_user", &elTable))
        {
            xdocLCUsers.pushCurrent(&elTable);
            Crs_lifeclinic_users rsLCUsers;
            if (!applyTable(xdocLCUsers, rsLCUsers, FLAG_INSERT | FLAG_UPDATE))
                throw E_FAIL;
            xdocLCUsers.popCurrent();
        }
    }
    catch(_com_error & e)
    {
        Error((BSTR) e.Description(), IID_IKCData, hr = e.Error());
    }
    catch(HRESULT hrError)
    {
        hr = hrError;
    }
    catch(...)
    {
        hr = E_FAIL;
    }

    return hr;
}


STDMETHODIMP CKCData::importData(VARIANT vFileName)
{
    HRESULT hr = S_OK;
    try
    {
        string strFileName = (char *) (_bstr_t) vFileName;

        CXmlDocument docTrans;
        if (!loadXmlFile(strFileName.c str(), docTrans, true))
            throw E_FAIL;

        CXmlElement elTable;

        // apply id map to the database
        if (docTrans.getItem("kc_id_map", &elTable))
        {
            docTrans.pushCurrent(&elTable);
            Crs_alternate_id rsIdMap;
            if (!applyTable(docTrans, rsIdMap, FLAG_INSERT))
                throw E_FAIL;
            docTrans.popCurrent();
        }
    }
    catch(_com_error & e)
    {
        Error((BSTR) e.Description(), IID_IKCData, hr = e.Error());
    }
    catch(HRESULT hrError)
    {
        hr = hrError;
    }
    catch(...)
    {
        hr = E_FAIL;
    }

    return hr;
}

STDMETHODIMP CKCData::checkKioskUserID(VARIANT vKioskID, VARIANT *pvIDExists)
{
    HRESULT hr = S_OK;
    bool fSuccess = false;

    try
    {

        Crs_kcdata rs;
        rs.setActiveCommand("cmdCheckIDExists");
        rs.setParameter("alternate_id", _variant_t(vKioskID));

        if (!m_pconn->execute(rs))
        {
            string strError;
            m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, hr = E_FAIL);
            throw hr;
        }

        if (!rs.isEmpty())
            fSuccess = true;
        else
            fSuccess = false;

    }
    catch(_com_error & e)
    {
        _bstr_t bstrError = e.Description();
        Error((char *) bstrError, IID_IKCData, hr = e.Error());
    }
    catch(HRESULT hrError)
    {
        hr = hrError;
    }
    catch(...)
    {
        hr = E_FAIL;
    }

    _variant_t vRetVal = fSuccess;
    *pvIDExists = vRetval;
    return hr;

}

bool CKCData::applyTable(CXmlDocument & xdoc, CSdoRecordset & rs, unsigned short
    fUpdateFlags)
{
    CXmlElement elTable;
    CXmlElement elRow;
    xdoc.getCurrent(&elTable);
    bool fRowFound = elTable.getFirst(&elRow);
    while(fRowFound)
    {
        // check if record exists
        xdoc.pushCurrent(&elRow);
        rs.setActiveCommand("applyExists");
        rs.getActiveCommand()->clearParms();
        rs.getActiveCommand()->setParmsOnly(xdoc);
        xdoc.popCurrent();
        if (!m_pconn->execute(rs))
        {
            string strError;
            m_pconn->getLastError(strError);
            Error(strError.c_str(), IID_IKCData, E_FAIL);
            return false;
        }
        bool fExists = !rs.isEmpty();
        rs.close();

        // if row doesn't exist and fInserts true then add it
        if (!fExists && (fUpdateFlags & FLAG_INSERT))
        {
            xdoc.pushCurrent(&elRow);
            rs.setActiveCommand("applyInsert");
            rs.getActiveCommand()->clearParms();
            rs.getActiveCommand()->setParmsOnly(xdoc);
            xdoc.popCurrent();
            if (!m_pconn->execute(rs))
            {
                string strError;
                m_pconn->getLastError(strError);
                Error(strError.c_str(), IID_IKCData, E_FAIL);
                return false;
            }
        }

        // if row exists and fUpdates true then update it
        if (fExists && (fUpdateFlags & FLAG_UPDATE))
        {
            xdoc.pushCurrent(&elRow);
            rs.setActiveCommand("applyUpdate");
            rs.getActiveCommand()->clearParms();
            rs.getActiveCommand()->setParmsOnly(xdoc);
            xdoc.popCurrent();
            if (!m_pconn->execute(rs))
            {
                string strError;
                m_pconn->getLastError(strError);
                Error(strError.c_str(), IID_IKCData, E_FAIL);
                return false;
            }
        }

        fRowFound = elTable.getNext(&elRow);
    }

    return true;
}

bool CKCData::loadXmlFile(LPCSTR pszFileName, CXmlDocument & xdocResult, bool
    fEncrypted)
{
    bool fSuccess = true;

    stringstream strmError;
    HRESULT hr = S_OK;

    try
    {
        strmError << "CKCData::loadXmlFile():";

        // open import file name
        FILE * pstream = fopen(pszFileName, "rb");
        if (pstream == NULL)
        {
            strmError << "Unable to open file [" << pszFileName << "].";
            Error(strmError.str().c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }

        // get the size of the file
        fseek(pstream, 0, SEEK_END);
        long lFileSize = ftell(pstream);
        fseek(pstream, 0, SEEK_SET);

        // allocate file buffer
        unsigned char * pBuff = new unsigned char [lFileSize + 1];
        if (pBuff == NULL)
        {
            strmError << "Unable to allocate file buffer. Out of memory.";
            Error(strmError.str().c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }
        pBuff[lFileSize] = 0;

        // read the import file
        long lBytesRead = fread(pBuff, 1, lFileSize, pstream);
        if (lBytesRead != lFileSize)
        {
            strmError << "CKCData::importData() failed. Unable to read file [";
            strmError << pszFileName << "].";
            Error(strmError.str().c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }

        // decrypt the file to xml
        string strXmlData;
        if (fEncrypted)
        {
            CEncryptor encrypt;
            encrypt.Decrypt((LPCSTR) pBuff, NULL, strXmlData);
        }
        else
            strXmlData = (const char *) pBuff;

        delete [] pBuff;

        // parse the xml
        xdocResult.loadDocument(strXmlData.c_str());
        if (!xdocResult.isReady())
        {
            string strParseError;
            xdocResult.getParserError(strParseError);
            strmError << "Unable to parse import file [" << pszFileName << "]. Error =✓
    [";
            strmError << strParseError << "].";
            Error(strmError.str().c_str(), IID_IKCData, E_FAIL);
            throw E_FAIL;
        }
    }
    catch(_com_error & e)
    {
        Error((BSTR) e.Description(), IID_IKCData, hr = e.Error());
        fSuccess = false;
    }
    catch(HRESULT hrError)
    {
        hr = hrError;
        fSuccess = false;
    }
    catch(...)
    {
        hr = E_FAIL;
        fSuccess = false;
    }

    return fSuccess;
}
```

```
// CoWeightReading.cpp : Implementation of CWeightReading
#include "stdafx.h"
#include "KCData.h"
#include "CoWeightReading.h"
#include "rs_kcdata.h"

/////////////////////////////////////////////////////////////////////////////
// CWeightReading

STDMETHODIMP CWeightReading::InterfaceSupportsErrorInfo(REFIID riid)
{
    static const IID* arr[] =
    {
        &IID_IWeightReading
    };
    for (int i=0; i < sizeof(arr) / sizeof(arr[0]); i++)
    {
        if (InlineIsEqualGUID(*arr[i],riid))
            return S_OK;
    }
    return S_FALSE;
}

HRESULT CWeightReading::FinalConstruct()
{
    m_vKioskUserID.vt = VT_NULL;
    m_vWeight.vt = VT_NULL;
    m_vReadingDate.vt = VT_NULL;

    return S_OK;
}

void CWeightReading::FinalRelease()
{
    return;
}


STDMETHODIMP CWeightReading::get_Weight(VARIANT *pVal)
{
    _variant_t vVal(*pVal, false);
    vVal = m_vWeight;
    *pVal = vVal.Detach();
    return S_OK;
}

STDMETHODIMP CWeightReading::put_Weight(VARIANT newVal)
{
    m_vWeight = newVal;
    return S_OK;
}

STDMETHODIMP CWeightReading::get_KioskUserID(VARIANT *pVal)
{
    _variant_t vVal(*pVal, false);
    vVal = m_vKioskUserID;
    *pVal = vVal.Detach();
    return S_OK;
}

STDMETHODIMP CWeightReading::get_ReadingDate(VARIANT *pVal)
{
    _variant_t vVal(*pVal, false);
    vVal = m_vReadingDate;
    *pVal = vVal.Detach();
    return S_OK;
}

STDMETHODIMP CWeightReading::put_ReadingDate(VARIANT newVal)
{
    m_vReadingDate = newVal;
    return S_OK;
}

/////////////////////////////////////////////////////////////////////////////
// internal C++ interface

bool CWeightReading::load(CSdoRecordset & rs)
{
    m_vWeight = rs.getField("weight");
    m_vReadingDate = rs.getField("reading_dt");
    m_vKioskUserID = rs.getField("kiosk_user_id");
    return true;
}


```

```cpp
// LCKioskServer.cpp : Implementation of WinMain


// Note: Proxy/Stub Information
//      To build a separate proxy/stub DLL,
//      run nmake -f LCKioskServerps.mk in the project directory.

#include "stdafx.h"
#include "resource.h"
#include <initguid.h>
#include "threadMain.h"
#include "LCKioskServer.h"
#include "registryKServer.h"

#include "LCKioskServer_i.c"


#include <stdio.h>

//////////////////////////////////////////////////////////////
// MFC support
CKioskServerApp _theApp;

///////////////////////////////////////////////////////////////
// ATL support
CServiceModule _Module;

//////////////////////////////////////////////////////////////
//Global decalrations
CLogNTEvents     _logEvents("Kiosk Server");
CLogFile         _logFile("c:\\LCKioskServer.log");
CLogDebug        _logDebug;
CLogMulti        _logAll;

BEGIN_OBJECT_MAP(ObjectMap)
END_OBJECT_MAP()


LPCTSTR FindOneOf(LPCTSTR p1, LPCTSTR p2)
{
    while (p1 != NULL && *p1 != NULL)
    {
        LPCTSTR p = p2;
        while (p != NULL && *p != NULL)
        {
            if (*p1 == *p)
                return CharNext(p1);
            p = CharNext(p);
        }
        p1 = CharNext(p1);
    }
    return NULL;
}

// Although some of these functions are big they are declared inline since they are
    only used once

inline HRESULT CServiceModule::RegisterServer(BOOL bRegTypeLib, BOOL bService)
{
    HRESULT hr = CoInitialize(NULL);
    if (FAILED(hr))
        return hr;

    // Remove any previous service since it may point to
    // the incorrect file
    Uninstall();

    // Add service entries
    UpdateRegistryFromResource(IDR_LCKioskServer, TRUE);

    // Adjust the AppID for Local Server or Service
    CRegKey keyAppID;
    LONG lRes = keyAppID.Open(HKEY_CLASSES_ROOT, _T("AppID"), KEY_WRITE);
    if (lRes != ERROR_SUCCESS)
        return lRes;

    CRegKey key;
    lRes = key.Open(keyAppID, _T("{BF823564-E93B-11D3-B88C-CC792E000000}"), KEY_WRITE)↙
    ;
    if (lRes != ERROR_SUCCESS)
        return lRes;
    key.DeleteValue(_T("LocalService"));

    if (bService)
    {
        key.SetValue(_T("LCKioskServer"), _T("LocalService"));
        key.SetValue(_T("-Service"), _T("ServiceParameters"));
        // Create service
        Install();
    }

    // Add object entries
    hr = CComModule::RegisterServer(bRegTypeLib);

    CoUninitialize();
    return hr;
}

inline HRESULT CServiceModule::UnregisterServer()
{
    HRESULT hr = CoInitialize(NULL);
    if (FAILED(hr))
        return hr;

    // Remove service entries
    UpdateRegistryFromResource(IDR_LCKioskServer, FALSE);
    // Remove service
    Uninstall();
    // Remove object entries
    CComModule::UnregisterServer(TRUE);
    CoUninitialize();
    return S_OK;
}

inline void CServiceModule::Init(_ATL_OBJMAP_ENTRY* p, HINSTANCE h, UINT                ↙
    nServiceNameID, const GUID* plibid)
{
    CComModule::Init(p, h, plibid);

    m_bService = TRUE;

    LoadString(h, nServiceNameID, m_szServiceName, sizeof(m_szServiceName) / sizeof    ↙
    (TCHAR));

    // set up the initial service status
    m_hServiceStatus = NULL;
    m_status.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
    m_status.dwCurrentState = SERVICE_STOPPED;
    m_status.dwControlsAccepted = SERVICE_ACCEPT_STOP;
    m_status.dwWin32ExitCode = 0;
    m_status.dwServiceSpecificExitCode = 0;
    m_status.dwCheckPoint = 0;
    m_status.dwWaitHint = 0;
}

LONG CServiceModule::Unlock()
{
    LONG l = CComModule::Unlock();
    if (l == 0 && !m_bService)
        PostThreadMessage(dwThreadID, WM_QUIT, 0, 0);
    return l;
}

BOOL CServiceModule::IsInstalled()
{
    BOOL bResult = FALSE;

    SC_HANDLE hSCM = ::OpenSCManager(NULL, NULL, SC_MANAGER_ALL_ACCESS);

    if (hSCM != NULL)
    {
        SC_HANDLE hService = ::OpenService(hSCM, m_szServiceName,
    SERVICE_QUERY_CONFIG);
        if (hService != NULL)
        {
            bResult = TRUE;
            ::CloseServiceHandle(hService);
        }
        ::CloseServiceHandle(hSCM);
    }
    return bResult;
}

inline BOOL CServiceModule::Install()
{
    if (IsInstalled())
        return TRUE;

    SC_HANDLE hSCM = ::OpenSCManager(NULL, NULL, SC_MANAGER_ALL_ACCESS);
    if (hSCM == NULL)
    {
        MessageBox(NULL, _T("Couldn't open service manager"), m_szServiceName, MB_OK);
        return FALSE;
    }

    // Get the executable file path
    TCHAR szFilePath[_MAX_PATH];
    ::GetModuleFileName(NULL, szFilePath, _MAX_PATH);

    SC_HANDLE hService = ::CreateService(
        hSCM, m_szServiceName, m_szServiceName,
        SERVICE_ALL_ACCESS, SERVICE_WIN32_OWN_PROCESS,
        SERVICE_DEMAND_START, SERVICE_ERROR_NORMAL,
        szFilePath, NULL, NULL, _T("RPCSS\0"), NULL, NULL);

    if (hService == NULL)
    {
        ::CloseServiceHandle(hSCM);
        MessageBox(NULL, _T("Couldn't create service"), m_szServiceName, MB_OK);
        return FALSE;
    }

    ::CloseServiceHandle(hService);
    ::CloseServiceHandle(hSCM);
    return TRUE;
}

inline BOOL CServiceModule::Uninstall()
```

```
{
    if (!IsInstalled())
        return TRUE;

    SC_HANDLE hSCM = ::OpenSCManager(NULL, NULL, SC_MANAGER_ALL_ACCESS);

    if (hSCM == NULL)
    {
        MessageBox(NULL, _T("Couldn't open service manager"), m_szServiceName, MB_OK);
        return FALSE;
    }

    SC_HANDLE hService = ::OpenService(hSCM, m_szServiceName, SERVICE_STOP | DELETE);

    if (hService == NULL)
    {
        ::CloseServiceHandle(hSCM);
        MessageBox(NULL, _T("Couldn't open service"), m_szServiceName, MB_OK);
        return FALSE;
    }
    SERVICE_STATUS status;
    ::ControlService(hService, SERVICE_CONTROL_STOP, &status);

    BOOL bDelete = ::DeleteService(hService);
    ::CloseServiceHandle(hService);
    ::CloseServiceHandle(hSCM);

    if (bDelete)
        return TRUE;

    MessageBox(NULL, _T("Service could not be deleted"), m_szServiceName, MB_OK);
    return FALSE;
}

/////////////////////////////////////////////////////////////////////////////
// Logging functions
void CServiceModule::LogEvent(LPCTSTR pFormat, ...)
{
    TCHAR chMsg[2048];
    va_list pArg;

    va_start(pArg,  pFormat);
    _vstprintf(chMsg, pFormat, pArg);
    va_end(pArg);

    CLogMsgEvent(LCEV_GENERIC, -1, chMsg).Post(_logAll);
}

////////////////////////////////////////////////////////////////////////////////////
// Service startup and registration
inline void CServiceModule::Start()
{
    SERVICE_TABLE_ENTRY st[] =
    {
        { m_szServiceName, _ServiceMain },
        { NULL, NULL }
    };
    if (m_bService && !::StartServiceCtrlDispatcher(st))
    {
        m_bService = FALSE;
    }
    if (m_bService == FALSE)
        Run();
}
```

```
inline void CServiceModule::ServiceMain(DWORD /* dwArgc */, LPTSTR* /* lpszArgv */)
{
    // Register the control request handler
    m_status.dwCurrentState = SERVICE_START_PENDING;
    m_hServiceStatus = RegisterServiceCtrlHandler(m_szServiceName, _Handler);
    if (m_hServiceStatus == NULL)
    {
        CLogMsgEvent("Handler not installed").Post(_logAll);
        return;
    }
    SetServiceStatus(SERVICE_START_PENDING);

    m_status.dwWin32ExitCode = S_OK;
    m_status.dwCheckPoint = 0;
    m_status.dwWaitHint = 0;

    // When the Run function returns, the service has stopped.
    Run();

    SetServiceStatus(SERVICE_STOPPED);
}

inline void CServiceModule::Handler(DWORD dwOpcode)
{
    switch (dwOpcode)
    {
    case SERVICE_CONTROL_STOP:
        SetServiceStatus(SERVICE_STOP_PENDING);
        PostThreadMessage(dwThreadID, WM_QUIT, 0, 0);
        break;
    case SERVICE_CONTROL_PAUSE:
        break;
    case SERVICE_CONTROL_CONTINUE:
        break;
    case SERVICE_CONTROL_INTERROGATE:
        break;
    case SERVICE_CONTROL_SHUTDOWN:
        break;
    default:
        CLogMsgEvent("Bad service request").Post(_logAll);
    }
}

void WINAPI CServiceModule::_ServiceMain(DWORD dwArgc, LPTSTR* lpszArgv)
{
    _Module.ServiceMain(dwArgc, lpszArgv);
}
void WINAPI CServiceModule::_Handler(DWORD dwOpcode)
{
    _Module.Handler(dwOpcode);
}

void CServiceModule::SetServiceStatus(DWORD dwState)
{
    m_status.dwCurrentState = dwState;
    ::SetServiceStatus(m_hServiceStatus, &m_status);
}

void CServiceModule::Run()
{
    _Module.dwThreadID = GetCurrentThreadId();

    HRESULT hr = CoInitializeEx(NULL, COINIT_MULTITHREADED);
    if (FAILED(hr))
    {
        CLogMsgEvent msg(LCEV_GENERIC, SVRTY_ERROR);
        msg << "CoInitializeEx() failed. Error = [0x" << std::hex << hr << "]";
        msg.Post(_logAll);
        return;
    }

    // This provides a NULL DACL which will allow access to everyone.
    CSecurityDescriptor sd;
    sd.InitializeFromThreadToken();
    hr = CoInitializeSecurity(sd, -1, NULL, NULL,
        RPC_C_AUTHN_LEVEL_PKT, RPC_C_IMP_LEVEL_IMPERSONATE, NULL, EOAC_NONE, NULL);
    _ASSERTE(SUCCEEDED(hr));

    hr = _Module.RegisterClassObjects(CLSCTX_LOCAL_SERVER | CLSCTX_REMOTE_SERVER, REGCLS_MULTIPLEUSE);
    _ASSERTE(SUCCEEDED(hr));

    ////////////////////////////////////////////////////////////////////
    // MFC support
    if (_theApp.InitApplication() == FALSE)
    {
        CLogMsgEvent msg(LCEV_GENERIC, SVRTY_ERROR);
        msg << "_theApp.InitApplication() failed";
        msg.Post(_logAll);
        return;
    }

    if (_theApp.InitInstance() == FALSE)
    {
        CLogMsgEvent msg(LCEV_GENERIC, SVRTY_ERROR);
        msg << "_theApp.InitInstance() failed";
        msg.Post(_logAll);
        _theApp.ExitInstance();
        return;
    }

    // end MFC support
    ////////////////////////////////////////////////////////////////////


    CLogMsgEvent("Service started").Post(_logAll);
    if (m_bService)
        SetServiceStatus(SERVICE_RUNNING);

    _theApp.Run();

    CLogMsgEvent("Service stopped").Post(_logAll);

    _Module.RevokeClassObjects();

    CoUninitialize();
}

/////////////////////////////////////////////////////////////////////////////
//
extern "C" int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine,
                                int nShowCmd)
{
    _logAll.AddLog(&_logEvents);
    _logDebug.Enabled(false);
    _logAll.AddLog(&_logFile);

#ifdef _DEBUG
    _logEvents.EnableTranslation(true);
    _logDebug.Enabled(true);
    _logAll.AddLog(&_logDebug);
#endif

    lpCmdLine = GetCommandLine(); //this line necessary for _ATL_MIN_CRT
    _Module.Init(ObjectMap, hInstance, IDS_SERVICENAME, &LIBID_LCKIOSKSERVERLib);
    _Module.m_bService = TRUE;

    TCHAR szTokens[] = _T("-/");

    LPCTSTR lpszToken = FindOneOf(lpCmdLine, szTokens);
    while (lpszToken != NULL)
    {
        if (lstrcmpi(lpszToken, _T("UnregServer"))==0)
            return _Module.UnregisterServer();

        // Register as Local Server
        if (lstrcmpi(lpszToken, _T("RegServer"))==0)
            return _Module.RegisterServer(TRUE, FALSE);

        // Register as Service
        if (lstrcmpi(lpszToken, _T("Service"))==0)
            return _Module.RegisterServer(TRUE, TRUE);

        // Initialize Configuration Registry Entries
        // Initialize Configuration Registry Entries
        if (lstrcmpi(lpszToken, _T("InitReg"))==0)
        {
            CRegistryKServer reg;
            reg.buildInitial();
            return 0;
        }

        lpszToken = FindOneOf(lpszToken, szTokens);
    }

    // Are we Service or Local Server
    CRegKey keyAppID;
    LONG lRes = keyAppID.Open(HKEY_CLASSES_ROOT, _T("AppID"), KEY_READ);
    if (lRes != ERROR_SUCCESS)
        return lRes;

    CRegKey key;
    lRes = key.Open(keyAppID, _T("{BF823564-E93B-11D3-B88C-CC792E000000}"), KEY_READ);
    if (lRes != ERROR_SUCCESS)
        return lRes;

    TCHAR szValue[_MAX_PATH];
    DWORD dwLen = _MAX_PATH;
    lRes = key.QueryValue(szValue, _T("LocalService"), &dwLen);

    _Module.m_bService = FALSE;
    if (lRes == ERROR_SUCCESS)
        _Module.m_bService = TRUE;

    // AFX internal initialization
    if (!AfxWinInit(hInstance, hPrevInstance, lpCmdLine, nShowCmd))
        CLogMsgEvent(LCEV_GENERIC, SVRTY_ERROR, "AfxWinInit failed.").Post(_logAll);
    else
        _Module.Start();

    // When we get here, the service has been stopped
    return _Module.m_status.dwWin32ExitCode;
}
```

```
// threadReceiver.cpp : implementation file
//

#include "stdafx.h"
#include "LCKioskServer.h"
#include "threadReceiver.h"
#include "filenameDelimited.h"
#include "threadMain.h"
#include "registryKServer.h"
#include "xmlKioskCmds.h"
#include "Encryptor.h"
#include "exports.h"
#include "brokerWrap.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CThreadReceiver

IMPLEMENT_DYNCREATE(CThreadReceiver, CThreadServer)

CThreadReceiver::CThreadReceiver()
{
    useTimers(m_pnTimers, TIMER_MAX);
}

CThreadReceiver::~CThreadReceiver()
{
}

BOOL CThreadReceiver::InitInstance()
{
    if (CThreadServer::InitInstance() == FALSE)
        return FALSE;

    m_fMaintenanceDone = false;
    return TRUE;
}

int CThreadReceiver::ExitInstance()
{
    // TODO:  perform any per-thread cleanup here
    return CThreadServer::ExitInstance();
}

BEGIN_MESSAGE_MAP(CThreadReceiver, CThreadServer)
    //{{AFX_MSG_MAP(CThreadReceiver)

    //}}AFX_MSG_MAP
    ON_THREAD_MESSAGE(WMUSER_START, onStart)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CThreadReceiver message handlers

LRESULT CThreadReceiver::onStart(WPARAM wParam, LPARAM lParam)
{
    CThreadServer::onStart(wParam, lParam);
    setTimer(TIMER_RECEIVE, 2000);

    // todo: testing
    setTimer(TIMER_CHECK_TIME, 1000);
    //setTimer(TIMER_CHECK_TIME, INTERVAL_CHECK_TIME);

    setTimer(TIMER_MOVE_FILES, INTERVAL_TIMER_MOVE_FILES);

    return FALSE;
}

void CThreadReceiver::onTimerIndex(int nIdx)
{
    killTimer(nIdx);

    SYSTEMTIME st;
    GetLocalTime(&st);
    SystemTimeToVariantTime(&st, &m_dateNow);

    switch (nIdx)
    {
    case TIMER_RECEIVE:
        onTimerReceiveFiles();
        break;
    case TIMER_CHECK_TIME:
        onTimerCheckTime();
        break;
    case TIMER_MOVE_FILES:
        onTimerMoveFiles();
        break;
    default:
        break;
    }

    return;
}

void CThreadReceiver::onTimerCheckTime()
{
    SYSTEMTIME  st;
    GetLocalTime(&st);
    COleDateTime odtNow;
    odtNow.SetTime(st.wHour, st.wMinute, st.wSecond);

    COleDateTimeSpan odtsNow = odtNow - _theApp.m_pregistry->m_odtEndOfDay;
    int nMinutes = odtsNow.GetTotalMinutes();
    if (nMinutes > 0 && nMinutes < 60)
    {
        if (!m_fMaintenanceDone)
        {
            m_fMaintenanceDone = true;
            deleteBackupFiles();
            checkLateKiosks();
            buildExports();
            buildUserExports();
        }
    }
    else
        m_fMaintenanceDone = false;

    setTimer(TIMER_CHECK_TIME, INTERVAL_CHECK_TIME);
}

void CThreadReceiver::onTimerReceiveFiles()
{
    CFileNameKiosk fnKiosk;
    CFileFind ffLocal;
    string strSearchFileName;
    CString cstrFileName;
```

[illegible]

```
    string strKioskId;
    string strWork;
    string strBackupFile;
    CFile  file;
    string strXmlCmd;
    string strXmlResult;
    string strXmlError;

    // set up wild card search name
    fnKiosk.set("direction", "U");
    fnKiosk.set("kiosk_id", "*");
    fnKiosk.set("date", "*");
    fnKiosk.setExtension("XML");
    fnKiosk.getFullName(strSearchFileName);

    // set current directory to the xfer directory
    _chdrive(_theApp.m_pregistry->m_lKioskDrive);
    string strDirectory = _theApp.m_pregistry->m_strKioskDirectory;
    strDirectory += "Xfer\\";
    _chdir(strDirectory.c_str());

    // look at each file in the xfer directory that matches wild card search
    bool fFileFound = ffLocal.FindFile(strSearchFileName.c_str()) != FALSE;
    while (fFileFound)
    {
        fFileFound = ffLocal.FindNextFile() != FALSE;
        cstrFileName = ffLocal.GetFileName();

        // parse filename and get out the kiosk id
        fnKiosk.setFullName(cstrFileName);
        fnKiosk.get("kiosk_id", strWork);
        strKioskId = strWork.substr(1, strWork.size() - 1); // drop K prefix

        // file errors can occur if we hit a file that is being xmitted, error is a ✔
    warning
        CFileException exFile;
        if (file.Open(cstrFileName, CFile::modeRead | CFile::shareExclusive, &exFile) ✔
    == FALSE)
        {
            char pszError [256];
            *pszError = 0;
            exFile.GetErrorMessage((LPSTR) pszError, 255);
            CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
            msg << "CThreadReceiver unable to open transaction file [";
            msg << (const char *) cstrFileName << "] for storage. Error = [";
            msg << pszError << "]. Cause = [" << exFile.m_cause << "]";
            msg.Post(_logAll);
            continue;
        }

        // read in the contents of the file. error is a warning.
        long lFileSize = file.GetLength();
        unsigned char * pBuff = new unsigned char [lFileSize + 1];
        pBuff[lFileSize] = 0;
        if (file.Read(pBuff, lFileSize) != lFileSize)
        {
            CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
            msg << "CThreadReceiver unable to read transaction file [";
            msg << (const char *) cstrFileName << "] for storage.";
            msg.Post(_logAll);
            file.Close();
            continue;
        }
        file.Close();

        // build xml command
        Cxdoc_uptKioskTrans docXmlCmd;
        docXmlCmd.setItemText("kiosk id", strKioskId.c str());
        docXmlCmd.setItemText("file_name", cstrFileName);

        // decrypt incomming data
        CEncryptor encrypt;
        string strTransXml;
        encrypt.Decrypt((LPCSTR) pBuff, NULL, strTransXml);
        delete [] pBuff;

        // parse incomming data
        CXmlDocument docXmlTrans(strTransXml.c_str());
        if (!docXmlTrans.isReady())
        {
            string strError;
            docXmlTrans.getParserError(strError);

            // event log message
            CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
            msg << "CThreadReceiver unable to parse transaction file [";
            msg << (const char *) cstrFileName << "]. XML Parser error - [";
            msg << strError << "]";
            msg.Post(_logAll);

            // alert personel message
            stringstream strmError;
            strmError << "The transaction file [" << (const char *) cstrFileName;
            strmError << "] could not be parsed. The error is [" << strError << "]. ";
            strmError << "You are listed as support personel for this kiosk [" <<
strKioskId;
            strmError << "]. Please investigate this problem ASAP.";

            Cxdoc_setKioskAlert docAlert;
            CXmlElement elParm;

            docAlert.getItem("parm", &elParm, Cxdoc_setKioskAlert::PARM_kiosk_id);
            elParm.setText(strKioskId.c_str());
            docAlert.getItem("parm", &elParm, Cxdoc_setKioskAlert::PARM_error_level);
            elParm.setText(2);
            docAlert.getItem("parm", &elParm, Cxdoc_setKioskAlert::PARM_subject);
            elParm.setText("Kiosk transaction file parse error");
            docAlert.getItem("parm", &elParm, Cxdoc_setKioskAlert::PARM_body);
            elParm.setText(strmError.str().c_str());
            docAlert.getXML(strXmlCmd);

            CBrokerWrap broker(m_spXmlCmds);
            if (!broker.execXml(strXmlCmd.c_str(), strXmlResult, strXmlError))
            {
                msg.Clear();
                msg << "CThreadReceiver unable to issue Kiosk alert.";
                msg.Post(_logAll);
            }

            continue;
        }

        CXmlElement elTrans;
        CXmlElement elData;

        docXmlTrans.getRoot(&elTrans);
        docXmlCmd.getItem("data", &elData);

        elData.addChild(&elTrans);

        docXmlCmd.getXML(strXmlCmd);

        // put it to the broker
        CBrokerWrap broker(m_spXmlCmds);
        if (!broker.execXml(strXmlCmd.c_str(), strXmlResult, strXmlError))
        {
            CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
            msg << "CThreadReceiver unable to store Kiosk transaction file [";
            msg << (const char *) cstrFileName << "].";
            msg.Post(_logAll);
            continue;
        }

        // move to backup directory
        strBackupFile = "..\\Backup\\";
        strBackupFile += (const char *) cstrFileName;
        if (rename(cstrFileName, strBackupFile.c_str()))
        {
            CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
            msg << "Unable to move file [" << (const char *) cstrFileName << "] to the
 ";
            msg << "backup directory. errno = [" << errno << "]";
            msg.Post(_logAll);
        }
    }

    setTimer(TIMER_RECEIVE, _theApp.m_pregistry->m_lReceiveFreqMs);
    return;
}

void CThreadReceiver::onTimerMoveFiles()
{
    CFileFind ffLocal;
    string strWorkDir = _theApp.m_pregistry->m_strKioskDirectory + "Work\\";
    string strXferDir = _theApp.m_pregistry->m_strKioskDirectory + "Xfer\\";

    string strSrcFile;
    string strDestFile;
    CString cstrFileName;

    // move files out to he xfer directory
    string strFileFind = strWorkDir + "*.*";
    BOOL fFileFound = ffLocal.FindFile(strFileFind.c_str());
    while (fFileFound)
    {
        fFileFound = ffLocal.FindNextFile();
        if (ffLocal.IsDirectory())
            continue;
        cstrFileName = ffLocal.GetFileName();
        strSrcFile = (LPCSTR) cstrFileName;
        strSrcFile.insert(0, strWorkDir);
        strDestFile = (LPCSTR) cstrFileName;
        strDestFile.insert(0, strXferDir);

        remove(strDestFile.c_str());
        rename(strSrcFile.c_str(), strDestFile.c_str());
    }

    // remove files marked as retrieved
    strFileFind = strXferDir + "*.*x";
    fFileFound = ffLocal.FindFile(strFileFind.c_str());
    while (fFileFound)
    {
        fFileFound = ffLocal.FindNextFile();
        if (ffLocal.IsDirectory())
            continue;
        cstrFileName = ffLocal.GetFilePath();
        remove(cstrFileName);
    }

    setTimer(TIMER_MOVE_FILES, INTERVAL_TIMER_MOVE_FILES);

    return;
}

bool CThreadReceiver::deleteBackupFiles()
{
    bool fSuccess = true;

    // delete backup file on disk
    string strFileName;
    string strSearchName("*.*");
    string strBackupDirectory = _theApp.m_pregistry->m_strKioskDirectory;
    strBackupDirectory += "Backup\\";
    strSearchName.insert(0, strBackupDirectory);

    // get current time - retentions days
    SYSTEMTIME  stime;
    GetLocalTime(&stime);
    COleDateTime odtKeep(stime);
    COleDateTimeSpan spanTime(_theApp.m_pregistry->m_lBackupRetentionDays, 0, 0, 0);
    odtKeep -= spanTime;

    // delete all file whose time is less than current time minus retention days
    CFileFind   ffLocal;
    bool fFileFound = ffLocal.FindFile(strSearchName.c_str()) != FALSE;
    while (fFileFound)
    {
        fFileFound = ffLocal.FindNextFile() != FALSE;
        if (ffLocal.IsDots())
            continue;

        FILETIME    ftime;
        ffLocal.GetCreationTime(&ftime);
        COleDateTime odtFile(ftime);
        if (odtFile < odtKeep)
        {
            CString cstrFileName = ffLocal.GetFilePath();
            remove(cstrFileName);
        }
    }

    // remove old daily transactions from database
    odtKeep.SetDate(stime.wYear, stime.wMonth, stime.wDay);
    spanTime.SetDateTimeSpan(_theApp.m_pregistry->m_lTransRetentionDays, 0, 0, 0);
    odtKeep -= spanTime;

    CXmlElement elDate;
    Cxdoc_deleteKioskTrans xdoc_deleteKioskTrans;
    xdoc_deleteKioskTrans.getItem("parm", &elDate);
    elDate.setText((LPCSTR) odtKeep.Format("%Y-%m-%d"));
    string strXmlCmd;
    string strXmlResult;
    string strXmlError;
    xdoc_deleteKioskTrans.getXML(strXmlCmd);
    CBrokerWrap broker(m_spXmlCmds);
    fSuccess = broker.execXml(strXmlCmd.c_str(), strXmlResult, strXmlError);

    return fSuccess;
}

bool CThreadReceiver::checkLateKiosks()
{
    Cxdoc_checkLateKiosks   docCmd;

    CXmlElement elParm;

    docCmd.getItem("parm", &elParm);
    elParm.setText(_theApp.m_pregistry->m_lHoursKioskAbsent);

    string strXmlCmd;
    string strXmlResult;
    string strXmlError;

    docCmd.getXML(strXmlCmd);

    CBrokerWrap broker(m_spXmlCmds);
    if (!broker.execXml(strXmlCmd.c_str(), strXmlResult, strXmlError))
    {
        CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
        msg << "CThreadReceiver unable to checkLateKiosks().";
        msg.Post(_logAll);
    }

    return true;
}

bool CThreadReceiver::buildExports()
{
    CExportKiosks ek(m_spXmlCmds);
    string strDir = _theApp.m_pregistry->m_strKioskDirectory + "Xfer\\";
    ek.setFinalDir(strDir.c_str());
    strDir = _theApp.m_pregistry->m_strKioskDirectory + "Work\\Kiosks\\";
    ek.setWorkDir(strDir.c_str());
    return ek.buildDailyExport();
}

bool CThreadReceiver::buildUserExports()
{
    string strDir;

    CExportLCUsers  eu(m_spXmlCmds);
    strDir = _theApp.m_pregistry->m_strKioskDirectory + "Work\\";
    eu.setFinalDir(strDir.c_str());
    strDir += "LCUsers\\";
    eu.setWorkDir(strDir.c_str());
    eu.setPacketSize(_theApp.m_pregistry->m_lLCUsersPerPacket);

    // build the big kahonee
    COleDateTime odt;
    odt.SetDate(1990, 12, 29);
    eu.buildDailyExport((DATE) odt, NULL, true);

    // build the daily
    odt = COleDateTime::GetCurrentTime();
    eu.buildDailyExport((DATE) odt, NULL, false);

    return true;
}


```

```
// wndMonitorISP.cpp : implementation file
//

#include "stdafx.h"
#include "lckioskclient.h"
#include "wndMonitorISP.h"
#include "filenameDelimited.h"
#include "zipUtil.h"
#include "Encryptor.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/*

   This window acts like the following state machine. A state change results in
   either a posted message are a set timer whose event will process that state.

    state           |   new state        |     condition
  ==================|====================|===========================================
  ST_CHECKTIME      | ST_CHECKTIME       | when it is not time to do anything
                    | ST_CONNECTING      | when it is time to process
  ------------------|--------------------|-------------------------------------------
  ST_CONNECTING     | ST_RETRY           | when the connect fails
                    | ST_EXCHANGE        | when the connect succeeds
                    | ST_CHECKTIME       | when connect fails and retries are exhausted
  ------------------|--------------------|-------------------------------------------
  ST_RETRY          | ST_CONNECTING      |
  ------------------|--------------------|-------------------------------------------
  ST_EXCHANGE       | ST_APPLY           | when the data exchange succeeds
                    | ST_RETRY           | when the data exchange fails
                    | ST_CHECKTIME       | when the data exchange fails and retries are
   exhausted
  ------------------|--------------------|-------------------------------------------
   ST_APPLY         | ST_CHECKTIME       | when apply updates succeeds
                    | ST_CHECKTIME       | when apply updates fails
*/

UINT CWndMonitorISP::m_nRasDialMsg = 0;

/////////////////////////////////////////////////////////////////////////////
// CWndMonitorISP

CWndMonitorISP::CWndMonitorISP()
{
    m_fUseRas = true;
    m_pdialer = NULL;
    m_pdialerRas = NULL;
    m_pdialerWinInet = NULL;
    m_fExchangeDone = false;
    m_fMaintenanceDone = false;
}

CWndMonitorISP::~CWndMonitorISP()
{
    if (m_pdialer != NULL)
        delete m_pdialer;
}

BEGIN_MESSAGE_MAP(CWndMonitorISP, CWndMonitor)
    //{{AFX_MSG_MAP(CWndMonitorISP)
    [illegible]
    ON_WM_TIMER()
    //}}AFX_MSG_MAP
    ON_MESSAGE(WMUSER_CONNECTED, onDialConnect)
    ON_MESSAGE(WMUSER_CHECKTIME, onTimerCheckTime)
    ON_MESSAGE(WMUSER_EXCHANGE, onExchange)
    ON_MESSAGE(WMUSER_APPLY, onApply)
    ON_MESSAGE(WMUSER_CONNECT, onTryConnect)
    ON_REGISTERED_MESSAGE(m_nRasDialMsg, onRasDialReport)
END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////////
// CWndMonitorISP message handlers

int CWndMonitorISP::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CWndMonitor::OnCreate(lpCreateStruct) == -1)
        return -1;

    buildDirectoryStructure();

    if (m_fUseRas)
    {
        m_pdialerRas = new CDialerRAS(m_registry.m_strPhoneBookEntry.c_str());
        m_nRasDialMsg = m_pdialerRas->getNotificationMessId();
        m_pdialer = m_pdialerRas;
    }
    else
    {
        m_pdialerWinInet = new CDialerWinInet();
        m_pdialer = m_pdialerWinInet;
    }

    m_pdialer->m_hwndOwner = m_hWnd;

    m_ePreviousState = ST_NONE;
    setState(ST_CHECKTIME);
    PostMessage(WMUSER_CHECKTIME);
    return 0;
}

void CWndMonitorISP::OnTimer(UINT nIDEvent)
{
    KillTimer(nIDEvent);

    switch(nIDEvent)
    {
    case TIMER_CHECKTIME:
        onTimerCheckTime(0, 0);
        break;
    case TIMER_RETRY:
        onTimerRetry();
        break;
    default:
        break;
    }
}

LRESULT CWndMonitorISP::onTimerCheckTime(WPARAM wParam, LPARAM lParam)
{
    COleDateTime odtNow = COleDateTime::GetCurrentTime();

    // check to see if it is end of day. If it is, then do maintenance
    COleDateTimeSpan odtsNow = odtNow - m_registry.m_odtEndOfDay;
    int nMinutes = odtsNow.GetTotalMinutes();

    if (nMinutes > 0 && nMinutes < 60)
    {
        if (!m_fMaintenanceDone)
        {
            deleteOldBackups();
            cleanChartersDirectory();
            return true;
        }
    }
    else
        m_fMaintenanceDone = false;

    // check to see if we should do file exchange
    odtsNow = odtNow - m_registry.m_odtTimeOfExchange;
    nMinutes = odtsNow.GetTotalMinutes();

    if (nMinutes >= 0)
    {
        // start the exchange
        if (m_registry.m_lConnectRetries)
            m_nRetryCount = m_registry.m_lConnectRetries;
        else
            m_nRetryCount = RETRY_FOREVER;

        setState(ST_CONNECTING);
        PostMessage(WMUSER_CONNECT);

        // set up the next exchange time
        COleDateTimeSpan odtsMinutes(0, 0, m_registry.m_lExchangeFreqMins, 0);
        m_registry.m_odtTimeOfExchange = odtNow + odtsMinutes;
    }
    else
    {
        setState(ST_CHECKTIME);
        SetTimer(TIMER_CHECKTIME, INTERVAL_CHECKTIME, NULL);
    }

    return 0;
}

bool CWndMonitorISP::onTimerRetry()
{
    if (m_nRetryCount > 0)
        m_nRetryCount--;

    CLogMsgEvent(LCEV_GENERIC, SVRTY_INFO, "Retrying connect").Post(_logAll);

    setState(ST_CONNECTING);
    PostMessage(WMUSER_CONNECT);

    return true;
}

LRESULT CWndMonitorISP::onTryConnect(WPARAM wParam, LPARAM lParam)
{
    if (m_registry.m_lNoDial != 0)
    {
        setState(ST_EXCHANGING);
        PostMessage(WMUSER_EXCHANGE);
    }
    else
        m_pdialer->connect();

    return 0;
}

LRESULT CWndMonitorISP::onExchange(WPARAM wParam, LPARAM lParam)[Claim 1b,4a,5a,5b,5c,
    7b,9b,10a]
{
    CLogMsgEvent(LCEV_GENERIC, SVRTY_INFO, "Exchanging files").Post(_logAll);

    bool fSuccess = exchangeFiles();

    m_pdialer->disconnect();

    if (fSuccess)
    {
        m_fMaintenanceDone = true;
        setState(ST_APPLYING);
        PostMessage(WMUSER_APPLY);
    }
    else
    {
        if (m_nRetryCount > 0 || m_nRetryCount == RETRY_FOREVER)
        {
            setState(ST_RETRYING);
            SetTimer(TIMER_RETRY, m_registry.m_lRetryIntervalSecs * 1000, NULL);
        }
        else
        {
            setState(ST_CHECKTIME);
            SetTimer(TIMER_CHECKTIME, INTERVAL_CHECKTIME, 0);
        }
    }

    return fSuccess ? TRUE : FALSE;
}

LRESULT CWndMonitorISP::onRasDialReport(WPARAM wRasConnState, LPARAM dwError)
{
    // if doing async Ras, logNotification must be called to recieve WMUSER_CONNECTED ↙
    message
    m_pdialerRas->logNotification(wRasConnState, dwError);
    return 0L;
}

LRESULT CWndMonitorISP::onDialConnect(WPARAM wParam, LPARAM dwError)
{
    if (dwError)
    {
        m_pdialer->disconnect();
        if (m_nRetryCount > 0 || m_nRetryCount == RETRY_FOREVER)
        {
            setState(ST_RETRYING);
            SetTimer(TIMER_RETRY, m_registry.m_lRetryIntervalSecs * 1000, NULL);
        }
        else
        {
            setState(ST_CHECKTIME);
            SetTimer(TIMER_CHECKTIME, INTERVAL_CHECKTIME, 0);
        }
    }
    else
    {
        setState(ST_EXCHANGING);
        PostMessage(WMUSER_EXCHANGE);
    }

    return 0;
}

bool CWndMonitorISP::_funcSortFileName(string & str1, string & str2)
{
    return str1.compare(str2) < 0;
}

bool CWndMonitorISP::applyLCUsersUpdates()[Claim 1f,1g]
{
    string strSearchName;
    string strFileName;
    string strBackupName;
    CString cstrZipFileName;
    CString cstrXmlFileName;

    CLogMsgEvent(LCEV_GENERIC, SVRTY_INFO, "Applying LCUsers.").Post(_logAll);

    bool fSuccess = true;

    try
    {
        string strProcDir = m_strProcDir + "LCUsers\\";

        strSearchName = "D_LCUsers*.zip";
        strSearchName.insert(0, m_strProcDir);

        CFileFind ffZips;
        BOOL fZipFound = ffZips.FindFile(strSearchName.c_str());
        while (fZipFound)
        {
            fZipFound = ffZips.FindNextFile();
            cstrZipFileName = ffZips.GetFilePath();
            clearDirectory(strProcDir.c_str());
            m_zipper.unzipFile(cstrZipFileName, strProcDir.c_str());


            strSearchName = strProcDir + "*.xml";
            CFileFind ffXmls;
            BOOL fXmlFound = ffXmls.FindFile(strSearchName.c_str());
            while (fXmlFound)
            {
                fXmlFound = ffXmls.FindNextFile();
                cstrXmlFileName = ffXmls.GetFilePath();
                m_spKCData->importLCUsers(_variant_t(cstrXmlFileName));
                remove(cstrXmlFileName);
            }

            strBackupName = m_strBackupDir + (LPCSTR) ffZips.GetFileName();
            rename(cstrZipFileName, strBackupName.c_str());
        }
    }
    catch(_com_error & e)
    {
        CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
        msg << "Error during applyLCUsersUpdates(). Error = [";
        msg.appendError(e);
        msg << "].";
        msg.Post(_logAll);
        fSuccess = false;
    }
    catch(CException * pE)
    {
        CLogMsgEvent msgError(LCEV_GENERIC, SVRTY_WARNING);
        char pszErrorMessage [256];
        pE->GetErrorMessage(pszErrorMessage, 256);
        msgError << "Error during applyLCUsersUpdates(). Error = [" << pszErrorMessage
    << "]";
        pE->Delete();
        fSuccess = false;
    }
    catch(...)
    {
        CLogMsgEvent(LCEV_GENERIC, SVRTY_WARNING, "Unknown exception during
    applyLCUsersUpdates().").Post(_logAll);
        fSuccess = false;
    }


    return fSuccess;
}

bool CWndMonitorISP::applyAppUpdates()
{
    string              strSearchName;
    string              strFileName;
    string              strBackupName;
    CString             cstrFileName;

    CLogMsgEvent        msgInfo(LCEV_GENERIC, SVRTY_INFO);

    msgInfo << "Applying application updates.";
    msgInfo.Post(_logAll);

    bool fSuccess = true;

    try
    {
        vector<string>  collFileNamesAll;
        vector<string>  collFileNamesById;

        /////////////////////////////////////////////////////////////////////
        // find application zip files
        /////////////////////////////////////////////////////////////////////
        // build wild card file name for KALL search
        CFileNameKiosk fnKiosk;
        fnKiosk.set("direction", "D");
        fnKiosk.set("kiosk_id", "KALL");
        fnKiosk.set("date", "*");
        fnKiosk.setExtension("zip");
        fnKiosk.getFullName(strSearchName);
        strSearchName.insert(0, m_strProcDir);

        // collect all KALL file name and sort them by date
        CFileFind ffLocal;
        BOOL fFileFound = ffLocal.FindFile(strSearchName.c_str());
        while (fFileFound)
        {
            fFileFound = ffLocal.FindNextFile();
            cstrFileName = ffLocal.GetFileName();
            collFileNamesAll.push_back((const char *) cstrFileName);
        }
        sort(collFileNamesAll.begin(), collFileNamesAll.end(), _funcSortFileName);

        // build wild card file name for K9999 search
        fnKiosk.set("kiosk_id", m_registry.m_strKioskId.c_str());
        fnKiosk.getFullName(strSearchName);
        strSearchName.insert(0, m_strProcDir);

        // collect all K9999 file names and sort them by date
        fFileFound = ffLocal.FindFile(strSearchName.c_str());
        while (fFileFound)
        {
            fFileFound = ffLocal.FindNextFile();
            cstrFileName = ffLocal.GetFileName();
            collFileNamesById.push_back((const char *) cstrFileName);
        }
        sort(collFileNamesById.begin(), collFileNamesById.end(), _funcSortFileName);

        // append K9999 filenames to the end of KALL filenames
        collFileNamesAll.insert(collFileNamesAll.end(), collFileNamesById.begin(), collFileNamesById.end());

        ////////////////////////////////////////////////////////////////////
        // apply application zip files
        ////////////////////////////////////////////////////////////////////
        vector<string>::iterator it;
        for (it = collFileNamesAll.begin(); it != collFileNamesAll.end(); it++)
        {
            strFileName = m_strProcDir + *it;
            if (!m_zipper.unzipFile(strFileName.c_str(), m_registry.m_strAppDirectory.c_str(),
                    CZipUtil::ZF_OverWrite | CZipUtil::ZF_UseDirectoryNames))
            {
                CLogMsgEvent msgError(LCEV_GENERIC, SVRTY_WARNING);
                msgError << "Unable to apply [" << *it << "]";
                msgError.Post(_logAll);
            }
            else
            {
                msgInfo.clear();
                msgInfo << "Applied update file [" << *it << "]";
                msgInfo.Post(_logAll);
            }

            strBackupName = m_strBackupDir + *it;
            strFileName = m_strProcDir + *it;
            if (rename(strFileName.c_str(), strBackupName.c_str()))
            {
                CLogMsgEvent msgError(LCEV_GENERIC, SVRTY_WARNING);
                msgError << "Unable to move [" << *it << "] to the backup directory after applying.";
                msgError << " errno - " << errno;
                msgError.Post(_logAll);
            }
        }
    }
    catch(_com_error & e)
    {
        CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
        msg << "Error during applyAppUpdates(). Error = [";
        msg.appendError(e);
        msg << "].";
        msg.Post(_logAll);
        fSuccess = false;
    }
    catch(CException * pE)
    {
        CLogMsgEvent msgError(LCEV_GENERIC, SVRTY_WARNING);
        char pszErrorMessage [256];
        pE->GetErrorMessage(pszErrorMessage, 256);
        msgError << "Error during applyAppUpdates(). Error = [" << pszErrorMessage << "]";
        pE->Delete();
        fSuccess = false;
    }
    catch(...)
    {
        CLogMsgEvent(LCEV_GENERIC, SVRTY_WARNING, "Unknown exception during applyAppUpdates().").Post(_logAll);
        fSuccess = false;
    }

    return fSuccess;
}

bool CWndMonitorISP::applyDBUpdates()
{
    string strSearchName;
    string strFileName;
    string strBackupName;
    CString cstrFileName;

    bool fSuccess = true;

    CLogMsgEvent(LCEV_GENERIC, SVRTY_INFO, "Applying database updates.").Post(_logAll);

    try
    {
        CFileNameKiosk fnKiosk;
        fnKiosk.set("direction", "D");
        fnKiosk.set("kiosk_id", m_registry.m_strKioskId.c_str());
        fnKiosk.set("date", "*");
        fnKiosk.setExtension("xml");
        fnKiosk.getFullName(strSearchName);
        strSearchName.insert(0, m_strProcDir);

        CFileFind ffLocal;
        BOOL fFileFound = ffLocal.FindFile(strSearchName.c_str());
        while (fFileFound)
        {
            fFileFound = ffLocal.FindNextFile();
            cstrFileName = ffLocal.GetFileName();
            strFileName = (LPCSTR) cstrFileName;
            strFileName.insert(0, m_strProcDir);
            m_spKCData->importData(_variant_t((LPCSTR) strFileName.c_str()));

            strBackupName = m_strBackupDir + (LPCSTR) cstrFileName;
            rename(strFileName.c_str(), strBackupName.c_str());
        }
    }
    catch(_com_error & e)
    {
        CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
        msg << "Error during applyDBUpdates(). Error = [";
        msg.appendError(e);
        msg << "].";
        msg.Post(_logAll);
        fSuccess = false;
    }
    catch(CException * pE)
    {
        CLogMsgEvent msgError(LCEV_GENERIC, SVRTY_WARNING);
        char pszErrorMessage [256];
        pE->GetErrorMessage(pszErrorMessage, 256);
        msgError << "Error during applyDBUpdates(). Error = [" << pszErrorMessage << "]";
        pE->Delete();
        fSuccess = false;
    }
    catch(...)
    {
        CLogMsgEvent(LCEV_GENERIC, SVRTY_WARNING, "Unknown exception during applyDBUpdates().").Post(_logAll);
        fSuccess = false;
    }

    return true;
```

```
}

LRESULT CWndMonitorISP::onApply(WPARAM, LPARAM)
{
    bool fSuccess = true;

    try
    {
        HRESULT hr = m_spKCData.CreateInstance(__uuidof(KCData));
        if (FAILED(hr))
        {
            CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
            msg << "Unable to instantiate IKCData. Error = [0x" << std::hex << hr << ↙
    "].";
            msg << " Database updates were not applied.";
            msg.Post(_logAll);
            throw E_FAIL;
        }

        m_spKCData->open();

        if (!m_zipper.createInstance())
            throw E_FAIL;

        m_strBackupDir = m_registry.m_strLocalDirectory + "Backup\\";
        m_strProcDir = m_registry.m_strLocalDirectory + "Process\\";

        applyDBUpdates();
        applyLCUsersUpdates();
        applyAppUpdates();

        m_spKCData->close();
    }
    catch(_com_error & e)
    {
        CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
        msg << "Error duringonApply(). Error = [";
        msg.appendError(e);
        msg << "].";
        msg.Post(_logAll);
        fSuccess = false;
    }
    catch(HRESULT hrError)
    {
        hrError;
        fSuccess = false;
    }

    // done, go back into a wait state till next maintenance run
    setState(ST_CHECKTIME);
    SetTimer(TIMER_CHECKTIME, INTERVAL_CHECKTIME, 0);

    return fSuccess ? TRUE : FALSE;
}

void CWndMonitorISP::clearDirectory(const char * pszDir)
{
    string strSearchName = pszDir;
    strSearchName += "*.*";

    CFileFind ffLocal;
    BOOL fFileFound = ffLocal.FindFile(strSearchName.c_str());
    while (fFileFound)
    {
        fFileFound = ffLocal.FindNextFile();
        if (ffLocal.IsDirectory())
```

```
            continue;
        CString cstrFileName = ffLocal.GetFilePath();
        remove(cstrFileName);
    }

    return;
}

bool CWndMonitorISP::cleanChartersDirectory()
{
    if (m_registry.m_strChartDir.size() == 0)
        return true;

    FILETIME ft;
    COleDateTime odtFile;
    COleDateTime odtYesterday = COleDateTime::GetCurrentTime();
    COleDateTimeSpan spanDay(1, 0, 0, 0);
    odtYesterday -= spanDay;

    CString cstrFileName;
    string strSearchName = m_registry.m_strChartDir + "*.*";

    CFileFind ffCharts;

    BOOL fFileFound = ffCharts.FindFile(strSearchName.c_str());
    while (fFileFound)
    {
        fFileFound = ffCharts.FindNextFile();
        if (ffCharts.IsDirectory())
            continue;
        ffCharts.GetLastWriteTime(&ft);
        odtFile = ft;
        if (odtFile < odtYesterday)
        {
            cstrFileName = ffCharts.GetFilePath();
            remove(cstrFileName);
        }
    }

    return true;
}

bool CWndMonitorISP::deleteOldBackups()
{
    bool fSuccess = true;

    // make the Backup directory current
    string strBackupDir = m_registry.m_strLocalDirectory + "Backup\\";
    _chdrive(m_registry.m_lLocalDrive);
    _chdir(strBackupDir.c_str());

    CFileFind               ffBackup;
    CString                 cstrFileName;
    CFileNameKiosk          fnKiosk;
    COleDateTime            odtDeleteDate;
    COleDateTimeSpan        spanRetentionPeriod;

    // set up the delete date by taking todays date and subtracting backup retention
days
    spanRetentionPeriod.SetDateTimeSpan(m_registry.m_lBackupRetentionDays, 0, 0, 0);
    SYSTEMTIME  st;
    GetLocalTime(&st);
    odtDeleteDate.SetDate(st.wYear, st.wMonth, st.wDay);
    odtDeleteDate -= spanRetentionPeriod;

    // set up wild card search
    fnKiosk.set("direction", "*");
    fnKiosk.set("kiosk_id", "*");
    fnKiosk.set("date", "*");
    fnKiosk.setExtension("*");
    string strSearch;
    fnKiosk.getFullName(strSearch);

    // retrieve list of files in backup directory
    bool fFileFound = ffBackup.FindFile(strSearch.c_str()) != FALSE;
    while (fFileFound)
    {
        fFileFound = ffBackup.FindNextFile() != FALSE;
        CString cstrFileName = ffBackup.GetFileName();
        fnKiosk.setFullName(cstrFileName);

        // retrieve the date portion of the file name
        DATE dateFile;
        fnKiosk.get("date", dateFile);
        COleDateTime odtFile(dateFile);

        // if the date falls before now minus retention period, then delete it
        if (odtFile < odtDeleteDate)
        {
            if (remove(cstrFileName))
            {
                CLogMsgEvent msgError(LCEV_GENERIC, SVRTY_WARNING);
                msgError << "Error deleting backup file. errno =  [" << errno << "]";
                fSuccess = false;
            }
        }
    }

    return fSuccess;
}

bool CWndMonitorISP::exchangeFiles()[Claim 1b,2a,4a,5a,6b,6c,7b,13a,13b]
{
    CLogMsgEvent msgError(LCEV_GENERIC, SVRTY_WARNING);
    msgError << "Error occured in exchangeFiles():";
    char * pszLastOperation;
    CFtpConnection * pconnFtp = NULL;

    bool fSuccess = false;

    try
    {
        ////////////////////////////////////////////////////////////////////////
        // ftp connect to the host
        ////////////////////////////////////////////////////////////////////////
        pszLastOperation = "CInternetSession():";
        CInternetSession sessionInternet(AfxGetAppName());

        pszLastOperation = "GetFtpConnection:";
        pconnFtp = sessionInternet.GetFtpConnection(
            m_registry.m_strFtpAddress.c_str(),
            m_registry.m_strFtpLogon.c_str(),
            m_registry.m_strFtpPassword.c_str());

        if (pconnFtp == NULL)
            throw E_FAIL;

        pszLastOperation = "SetCurrentDirectory():";
        if (!pconnFtp->SetCurrentDirectory(m_registry.m_strHostDirectory.c_str()))
            throw E_FAIL;

        ////////////////////////////////////////////////////////////////////////
        // get files peculiar to this kiosk
        //////////////////////////////////////////////////////////////////////////
        CFileNameKiosk fnKiosk;
        fnKiosk.set("direction", "D");
        fnKiosk.set("kiosk_id", m_registry.m_strKioskId.c_str());
        fnKiosk.set("date", "*");
        fnKiosk.setExtension("zip");
        string strSearchFileName;
        fnKiosk.getFullName(strSearchFileName);
        pszLastOperation = "pullFtpFiles(KioskId.zip):";
        pullFtpFiles(pconnFtp, strSearchFileName.c_str(), true);

        fnKiosk.setExtension("xml");
        fnKiosk.getFullName(strSearchFileName);
        pszLastOperation = "pullFtpFiles(KioskId.xml):";
        pullFtpFiles(pconnFtp, strSearchFileName.c_str(), true);

        //////////////////////////////////////////////////////////////////////////
        // get application files targeted for all kiosk
        //////////////////////////////////////////////////////////////////////////
        fnKiosk.set("kiosk_id", "KALL");
        fnKiosk.setExtension("zip");
        fnKiosk.getFullName(strSearchFileName);
        pszLastOperation = "pullFtpFiles(KALL.zip):";
        pullFtpFiles(pconnFtp, strSearchFileName.c_str(), false);

        //////////////////////////////////////////////////////////////////////////
        // get lifeclinic users
        //////////////////////////////////////////////////////////////////////////
        CFileNameDelimited fnLCUsers;
        fnLCUsers.append("direction", "D");
        fnLCUsers.append("file", "LCUSERS");
        fnLCUsers.append("date", "*");
        fnLCUsers.setExtension("zip");
        fnLCUsers.getFullName(strSearchFileName);
        pszLastOperation = "pullFtpFiles(LCUsers):";
        pullFtpFiles(pconnFtp, strSearchFileName.c_str(), false);

        //////////////////////////////////////////////////////////////////////////
        // put daily transaction files to the host
        //////////////////////////////////////////////////////////////////////////
        pszLastOperation = "ExportData:";
        exportData();

        // set up wild card search name
        fnKiosk.set("direction", "U");
        fnKiosk.set("kiosk_id", m_registry.m_strKioskId.c_str());
        fnKiosk.set("date", "*");
        fnKiosk.setExtension("xml");
        fnKiosk.getFullName(strSearchFileName);
        pszLastOperation = "pushFtpFiles():";
        pushFtpFiles(pconnFtp, strSearchFileName.c_str());
        fSuccess = true;
    }
    catch(CInternetException * pE)
    {
        char pszErrorMessage [256];
        pE->GetErrorMessage(pszErrorMessage, 256);
        msgError << pszLastOperation << " Error = [0x" << std::hex << pE->m_dwError;
        msgError << "(" << std::dec << pE->m_dwError << ")]. ";
        msgError << "Error message = [" << pszErrorMessage << "]";
        pE->Delete();
    }
    catch(CException * pE)
    {
        char pszErrorMessage [256];
        pE->GetErrorMessage(pszErrorMessage, 256);
        msgError << pszLastOperation << "Error message = [" << pszErrorMessage << "]";
        pE->Delete();
    }
    catch(DWORD dwError)
    {
        msgError << pszLastOperation << "Error = [0x" << std::hex << dwError;
        msgError << "(" << std::dec << dwError << ")]. ";
    }
    catch(...)
    {
        msgError << pszLastOperation << "Error = [Unknown Exception]";
    }

    if (pconnFtp != NULL)
    {
        pconnFtp->Close();
        delete pconnFtp;
    }

    if (!fSuccess)
        msgError.Post(_logAll);

    return fSuccess;
}

bool CWndMonitorISP::pullFtpFiles(CFtpConnection * pconnFtp, LPCSTR pszFileSearch,
    bool fMarkHostFile)[Claim 1b,2a,4a,6c,7b]
{
    CString         cstrFileName;
    string          strLocalFileName;
    string          strProcessFileName;
    CFileFind       ffLocal;

    string strXferDir = m_registry.m_strLocalDirectory + "Xfer\\";
    string strBackupDir = m_registry.m_strLocalDirectory + "Backup\\";
    string strProcDir = m_registry.m_strLocalDirectory + "Process\\";

    CFtpFileFind    ffHost(pconnFtp);

    // get the directory
    BOOL fFileFound = ffHost.FindFile(pszFileSearch);

    // now get each file in list
    while (fFileFound)
    {
        fFileFound = ffHost.FindNextFile();
        cstrFileName = ffHost.GetFileName();

        // check to see if we've already pulled or applied file, if so, don't pull it
        strLocalFileName = strBackupDir + (const char *) cstrFileName;
        if (ffLocal.FindFile(strLocalFileName.c_str()))
            continue;

        strLocalFileName = strProcDir + (const char *) cstrFileName;
        if (ffLocal.FindFile(strLocalFileName.c_str()))
            continue;

        // pull file over ftp
        strLocalFileName = strXferDir + (const char *) cstrFileName;
        if (!pconnFtp->GetFile(cstrFileName, strLocalFileName.c_str()))
            throw GetLastError();

        // move the file to the process directory
        strProcessFileName = strProcDir + (const char *) cstrFileName;
        if (rename(strLocalFileName.c_str(), strProcessFileName.c_str()))
```

```
            throw (DWORD) errno;

        if (fMarkHostFile)
        {
            // rename file on host, deletes do not work on cached ftp files
            string strNewName = (const char *)cstrFileName;
            strNewName += 'x';
            if (!pconnFtp->Rename(cstrFileName, strNewName.c_str()))
            {
                DWORD dwError = GetLastError();
                CLogMsgEvent msgError(LCEV_GENERIC, SVRTY_WARNING);
                msgError << "Unable to mark host file [";
                msgError << (const char *) cstrFileName << "]. Error = [0x";
                msgError << std::hex << dwError << std::dec << " (" << dwError << ") ]↙
    .";
                msgError.Post(_logAll);
            }
        }
    }

    return true;
}

bool CWndMonitorISP::pushFtpFiles(CFtpConnection * pconnFtp, LPCSTR pszFileSearch)
{
    CString cstrFileName;
    string strLocalFileName;
    string strXferDir = m_registry.m_strLocalDirectory + "Xfer\\";
    string strBackupDir = m_registry.m_strLocalDirectory + "Backup\\";
    string strSearchFileName = strXferDir + pszFileSearch;

    CFileFind ffLocal;
    BOOL fFileFound = ffLocal.FindFile(strSearchFileName.c_str());
    while(fFileFound)
    {
        fFileFound = ffLocal.FindNextFile();
        cstrFileName = ffLocal.GetFileName();

        // send the file
        strLocalFileName = strXferDir + (const char *) cstrFileName;
        if (!pconnFtp->PutFile(strLocalFileName.c_str(), cstrFileName))
            throw GetLastError();

        // move the file to the backup directory
        string strBackupFileName = strBackupDir + (const char *) cstrFileName;
        if (rename(strLocalFileName.c_str(), strBackupFileName.c_str()))
        {
            CLogMsgEvent msgError(LCEV_GENERIC, SVRTY_WARNING);
            msgError << "Unable to move file [" << (const char *) cstrFileName << "] "↙
    ;
            msgError << "to the backup directory.";
            msgError.Post(_logAll);
        }
    }
    return true;
}

bool CWndMonitorISP::exportData()[Claim 5a]
{
    bool fSuccess = true;

    string strFileName;
    SYSTEMTIME tm;
    GetLocalTime(&tm);
    DATE dateNow;

    // build file name
    SystemTimeToVariantTime(&tm, &dateNow);
    CFileNameKiosk fnKiosk;
    fnKiosk.set("direction", "U");
    fnKiosk.set("kiosk_id", m_registry.m_strKioskId.c_str());
    fnKiosk.set("date", dateNow);
    fnKiosk.setExtension("xml");
    fnKiosk.getFullName(strFileName);

    // get the exported data
    IKCDataPtr spKCData;
    HRESULT hr = spKCData.CreateInstance(__uuidof(KCData));
    if (FAILED(hr))
    {
        CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
        msg << "Unable to instantiate IKCData object. Error = [0x";
        msg << std::hex << hr << "].";
        msg.Post(_logAll);
    }

    try
    {
        // get the exported data
        spKCData->open();
        _variant_t vData = spKCData->getUnexportedData();
        string strXml = (char *) (_bstr_t) vData;

        // encrypt the data
        CEncryptor encrypt;
        string strXmlEncrypted;
        encrypt.Encrypt(strXml.c_str(), NULL, strXmlEncrypted);

        // write the data out
        CFile fileOut;
        string strFileOut = m_registry.m_strLocalDirectory + "Xfer\\";
        strFileOut += strFileName;
        fileOut.Open(strFileOut.c_str(), CFile::modeCreate | CFile::modeWrite);
        fileOut.Write(strXmlEncrypted.c_str(), strXmlEncrypted.size());
        fileOut.Close();

        // mark all data exported
        spKCData->markDataExported();

        spKCData->close();

        fSuccess = true;
    }
    catch(_com_error & e)
    {
        CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
        msg.setError(e);
        msg.Post(_logAll);
        fSuccess = false;
    }
    catch(CFileException * pEx)
    {
        char szError [256];
        pEx->GetErrorMessage(szError, 256);
        CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
        msg << "CFile threw exception. Error = [" << szError << "]. ";
        msg << "Cause = [" << pEx->m_cause << "].";
        msg.Post(_logAll);
        pEx->Delete();
    }
    catch(bool fError)
    {
            fSuccess = fError;
    }
    catch(...)
    {
        CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
        msg << "Unknown exception type caught in CWndMonitorISP::exportData().";
        msg.Post(_logAll);
        fSuccess = false;
    }

    return fSuccess;
}

void CWndMonitorISP::buildDirectoryStructure()
{
    string strBaseDir = m_registry.m_strLocalDirectory;
    string strDir = strBaseDir + "Xfer\\";
    mkdir(strDir.c_str());
    strDir = strBaseDir + "Backup\\";
    mkdir(strDir.c_str());
    strDir = strBaseDir + "Process\\";
    mkdir(strDir.c_str());
    strDir += "LCUsers\\";
    mkdir(strDir.c_str());
    return;
}
```

```
#include "xc_OtherCommands.h"
#include "rs_kiosk.h"

///////////////////////////////////////////////////////////////////////
CXC_IMPLEMENT_FACTORY(Cxc_applyKioskTrans)

bool Cxc_applyKioskTrans::execCommand()
{
    bool fSuccess = true;
    bool fTransStarted = false;
    m_pconn   = NULL;
    m_lAuditId = 0;

    try
    {
        Crs_kiosk_daily_trans   rsTrans;

        string strKioskId;
        string strRecId;

        getParm("kiosk_id", strKioskId);
        getParm("rec_id", strRecId);

        long lKioskId = atol(strKioskId.c_str());
        long lRecId = atol(strRecId.c_str());

        if (lKioskId == 0)
        {
            m_emLast << "\"kiosk_id\" is a required parameter.";
            throw fSuccess = false;
        }

        if (lRecId == 0)
        {
            m_emLast << "\"rec_id\" is a required parameter.";
            throw fSuccess = false;
        }

        m_pconn = m_pcoClient->getConnection();
        if (m_pconn == NULL)
        {
            m_emLast.setError(m_pcoClient->getLastError());
            throw fSuccess = false;
        }

        rsTrans.setActiveCommand("cmdGetTrans");
        rsTrans.setParameter("kiosk_id", _variant_t(lKioskId));
        rsTrans.setParameter("rec_id", _variant_t(lRecId));

        if (!m_pconn->execute(rsTrans))
        {
            m_emLast.setError(m_pconn->getLastError());
            throw fSuccess = false;
        }

        if (rsTrans.isEmpty())
        {
            m_emLast << "There are no transactions for kiosk_id [" << lKioskId << "], ↙
rec_id [";
            m_emLast << lRecId << "].";
            throw fSuccess = false;
        }

        // get the transactions in xml format
        string strXml;
        rsTrans.getField("data", strXml);
    CXmlDocument xdocTrans(strXml.c_str());
    if (!xdocTrans.isReady())
    {
        string strParseError;
        xdocTrans.getParserError(strParseError);
        m_emLast << "Unable to parse XML transactions. kiosk_id = [" << lKioskId;
        m_emLast << "], rec_id [" << lRecId << "]. Error = [" << strParseError << ↙
"].";
        throw fSuccess = false;
    }

    CXmlElement elTable;

    fTransStarted = m_pconn->beginTrans();

    m_lAuditId = getAuditId();

    // apply alternate id
    if (xdocTrans.getItem("kc_id_map", &elTable))
    {
        xdocTrans.pushCurrent(&elTable);
        Crs_kc_id_map rsAId;
        rsAId.setActiveCommand("applyTrans");
        fSuccess = applyTransactions(xdocTrans, rsAId);
        xdocTrans.popCurrent();
    }

    // apply kiosk users
    if (fSuccess && xdocTrans.getItem("kc_user", &elTable))
    {
        xdocTrans.pushCurrent(&elTable);
        Crs_kc_user rsUser;
        rsUser.setActiveCommand("applyTrans");
        fSuccess = applyTransactions(xdocTrans, rsUser);
        xdocTrans.popCurrent();
    }

    // apply blood pressures
    if (fSuccess && xdocTrans.getItem("kc_blood_pressure", &elTable))
    {
        xdocTrans.pushCurrent(&elTable);
        Crs_kc_blood_pressure rsBP;
        rsBP.setActiveCommand("applyTrans");
        fSuccess = applyTransactions(xdocTrans, rsBP);
        xdocTrans.popCurrent();
    }

    // apply weights
    if (fSuccess && xdocTrans.getItem("kc_weight", &elTable))
    {
        xdocTrans.pushCurrent(&elTable);
        Crs_kc_weight rsWeights;
        rsWeights.setActiveCommand("applyTrans");
        fSuccess = applyTransactions(xdocTrans, rsWeights);
        xdocTrans.popCurrent();
    }

    fTransStarted = false;

    string strStatus;
    string strReason;
    if (fSuccess)
    {
        m_pconn->commitTrans();
        strStatus = "P";
    }
```

```
        else
        {
            m_pconn->rollbackTrans();
            strStatus = 'E';
            m_emLast.getError(strReason);
        }

        // mark transaction processed
        rsTrans.setActiveCommand("cmdUptKioskTrans");
        rsTrans.setParameter("kiosk_id", _variant_t(lKioskId));
        rsTrans.setParameter("rec_id", _variant_t(lRecId));
        rsTrans.setParameter("status", _variant_t(strStatus.c_str()));
        rsTrans.setParameter("reason", _variant_t(strReason.c_str()));
        rsTrans.setParameter("audit_id", _variant_t(1L));
        if (!m_pconn->execute(rsTrans))
        {
            m_emLast.setError(m_pconn->getLastError());
            throw fSuccess = false;
        }
    }
    catch(_com_error & e)
    {
        m_emLast.setError(e);
        fSuccess = false;
    }
    catch(bool fBool)
    {
        fBool;
        fSuccess = false;
    }
    catch(...)
    {
        m_emLast.setError("Unkown exception raised. [Command:applyKioskTrans]");
        fSuccess = false;
    }

    if (fTransStarted)
    {
        if (fSuccess)
            m_pconn->commitTrans();
        else
            m_pconn->rollbackTrans();
    }

    return fSuccess;
}

bool Cxc_applyKioskTrans::applyTransactions(CXmlDocument &xdocData, CSdoRecordset &
    rsCmdRecSet)
{
    bool fSuccess = false;
    string strTag;
    m_emLast.clear();

    try
    {
        //Active  command is expected to be set prior calling.

        //get the Sdo command pointer
        CSdoCommand * pcmdSdo = rsCmdRecSet.getActiveCommand();

        //get the table element from the stack
        CXmlElement elTable;
        xdocData.getCurrent(&elTable);
        string strAttrValue;
        elTable.getAttribute("t", strAttrValue);
        if (strAttrValue[0] != 't')
        {
            m_emLast.clear();
            m_emLast << "Expecting XML table element to have attribute t=\"t\".";
            throw fSuccess = false;
        }

        CXmlElement elRow;
        bool fRows = elTable.getFirst(&elRow);  //get the "row" Element

        while (fRows)                           //process for all "row" Elements
        {
            //get "row"tag
            elRow.getTag(strTag);
            if (stricmp(strTag.c_str(), "row") != 0)
            {
                m_emLast.clear();
                m_emLast << "Expecting XML element \"row\". Found \"" << strTag << "\"↙
    .";
                fSuccess = false;
                break;
            }

            //Set all the parameters
            xdocData.pushCurrent(&elRow);

            pcmdSdo->clearParms();
            if (pcmdSdo->setParms(xdocData) ==  false)
            {
                string strError;
                pcmdSdo->getLastError(strError);
                m_emLast.setError(strError.c_str());
                fSuccess = false;
                break;
            }

            if (m_lAuditId)
                pcmdSdo->setParm("audit_id", _variant_t(m_lAuditId));

            xdocData.popCurrent();

            // update the table
            if (!(fSuccess = m_pconn->execute(rsCmdRecSet)))
            {
                m_emLast.setError(m_pconn->getLastError());
                fSuccess = false;
                break;
            }

            //fetch next row to update.
            fRows = elTable.getNext(&elRow);
        }

        fSuccess = true;
    }
    catch(_com_error & e)
    {
        m_emLast.setError(e);
        fSuccess = false;
    }
    catch(bool fBool)
    {
        fBool;
        fSuccess = false;
    }
    catch(...)
    {
        m_emLast.setError("Unkown exception raised.[CxcLCBrokerModify::execute ↙
(parameters)]");
        fSuccess = false;
    }

    return fSuccess;
}
```